Interpreted Dependency Networks:

A General Framework for Belief Revision

by

Guy R.E.S. Saward

This thesis is submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

July 1991

Supervisors:
Dr S.B.Russ
Dr A. G. Cohn (University of Leeds)

Department of Computer Science
University of Warwick

Abstract

Belief Revision is a fundamental component of intelligent behaviour and is therefore an area of study within Artificial Intelligence (AI). In many cases the belief revision is the result of modifying a long term model of some domain by the addition of information specific to particular instances of a problem within that domain. This thesis argues the case for a *Foundations* approach to belief revision in which the level of belief in any proposition is supported by explicit justifications. Networks of such justifications can be used as long term knowledge stores thereby capturing the dependencies between different pieces of information.

Truth Maintenance Systems (TMSs), also known as Reason Maintenance Systems, are a class of programme that provides the functionality necessary to perform belief revision in just this way. However, each individual style of TMS contains embedded design decisions based on a particular problem solving domain and/or approach. An *Interpreted Dependency Network* (IDN) is the embodiment of the philosophy behind TMSs without the built-in assumptions. As such, IDNs allow for the easy specification of belief revision systems. Both the generality of IDNs and the ease of specification will be shown by rationally reconstructing existing approaches to TMSs.

This thesis provides a declarative semantics for IDNs along with general purpose algorithms for interpreting dependency networks. This means that IDNs have the necessary support to function as representations for long term knowledge. This is in contrast to the transitory nature of the information normally captured by TMSs.

The ability of IDNs to be used as knowledge representations is demonstrated by reconstructing several existing AI representation paradigms as IDNs. This also highlights the benefits of IDNs. For example, the declarative nature of the interpretation mechanism enables easy specification of the semantics of a representation while the explicit representation of dependencies enables a network to support several different modes of reasoning.

Acknowledgements

In the time honoured way of all theses, I would like to thank my supervisor Tony Cohn for all his criticism, support and encouragement (in order of increasing supply) over the last four years. In the same vein, I would also like to thank Steve Russ for his boundless bonhomie and ceaseless pestering over the last few months that have preceded this submission. Thanks must also go to the department of Computer Science for their interest in my progress and the use of the departmental facilities.

For their financial support, I must acknowledge the role played by SERC. Without their studentship I would not have been able to start, let alone finish, my studies.

To the other runners in the great WAIB thesis race, I can only offer my congratulations to those who have already passed the post, and my encouragement to all those yet to go the final furlong. I am indebted to all of them for their friendship, thoughts and time. In running order, I would like to pay homage to Kathy Courtney, Harjinda Chayra, Mike Slade and David Randel, and I would like to crack the whip at Ian Gent and Felix Hovsepian.

Saving the best until last, I must thank my girlfriend, fiancee and wife for all her patience, tolerance and love. Without Liz Saward (nee Tregoning) this thesis would never have been finished. In addition, a special (posthumous) dedication must go to all the brain cells that died so that this thesis could be proof read!

Declaration

I declare that everything contained in this thesis, unless otherwise acknowledged, is my own work.

Contents

Figures

CHAPTER 1

Introduction

Most of science and much of everyday "common sense" reasoning uses abstract models of reality to reason about the world, be it the interpretation of observed phenomena, the prediction of future events or the planning of future events. Computers, by their very nature, are excellent tools for manipulating numeric models that operate using well-defined algorithms. The term Artificial Intelligence (AI) means many things to many people but there are characteristics that distinguish AI from other areas of computation, these being the type of domain being modelled, how the model is constructed and how it is used.

Depending on one's view of Artificial Intelligence, the long-term goal of AI is the creation or modelling of "intelligent" computers. This means that domains to which AI is applied are those in which an agent is deemed to require significant amounts of knowledge or expertise if they are to operate successfully. The models that are constructed in AI are not numeric but symbolic. Although numeric models may contain concepts such as forces, bank balances or geometric information, the meaning is implicit in the model. AI deals directly (but not necessarily exclusively) with such symbolic concepts. This is a result (and/or cause) of the kinds of domains being tackled where the system that is being modelled is not a physical system, but is itself a model either of some physical domain or some intellectual activity, be it an expert's understanding of how to diagnose diseases or an attempt to capture planning processes.

The need to represent symbolic knowledge has led to the development of a wide range of systems ranging from semantic networks and inheritance systems, through frame-based systems and production rules, to logic in all its various forms. The goal of all these systems is to capture some set of facts and the relationships between them. The information that is captured in this way can be divided into explicit and implicit knowledge, with implicit information being constructed from explicit information using some reasoning or inference method.

Any logical system demonstrates this distinction very well. Given a set of axioms, the explicit information, a proof theory can be used to derive implicit consequences of those axioms. Thus, given axioms $\{\alpha, \alpha \rightarrow \beta\}$ in classical logic, the consequent β can be derived. However, β being true was implicit in the set of axioms for any model that satisfied α and

$\alpha \rightarrow \beta$ would also satisfy β. There are problems with this approach to knowledge representation for it concentrates on the derivation of implicit information from explicit information to the detriment of exploring the consequences of changes in the explicit information.

1.1 Belief Revision

The interaction between a logical theory and the system being modelled can be more complex than simply having an explicit set of axioms to model a particular system and then applying some procedure to elicit the implicit information. The set of axioms may alter to reflect changes in the system over time, or the observation of more information, or the discovery of previous errors in the model or observations. Therefore, I wish to draw a distinction between logical truths (tautologies) and a set of axioms (or assumptions) which may change and which I will call "beliefs". Furthermore, we cannot talk about the truth of a derived proposition, unless it is a tautology, but merely in our belief or disbelief in it, given a particular set of assumptions. As the set of assumptions changes so does the belief or disbelief of information derived from those assumptions.

In this thesis I propose that knowledge in AI domains subject to such changes should be captured in *Dependency Networks*: a directed graph in which nodes represent propositions; and arcs represent inferences capable of deriving or supporting a level of belief in one node given a particular level of belief in another. Each proposition should then have a level of belief that reflects the current support for it, i.e. the validity of assumptions and inferences from which the proposition can be derived.

The body of the thesis is devoted to describing a system for both labelling propositions with levels of beliefs and maintaining that labelling as the set of assumptions changes. This represents a generalisation of existing Truth Maintenance Systems (also known as Reason Maintenance Systems). In addition to rationally reconstructing such systems, I look at the complexity involved in constructing and maintaining such labellings in relation to the properties of the rules for combining beliefs and the topology of network. This approach to belief revision is then applied to several existing knowledge representations and reasoning

paradigms to gain insight into their behaviour and to develop a rich interaction between system and user capable of supporting a wide range of tasks.

The rest of this chapter motivates ideas contained in the formulation of Interpreted Dependency Networks (IDNs). Section 1.2 looks at some existing areas of AI from the perspective of belief revision and examines their usefulness as belief revision mechanisms. This serves three purposes:

- it highlights the limitations of such approaches (§1.2.1-§1.2.4) as non-monotonic logics, consistency-based approaches to belief revision, and lattice-based approaches;
- it generates a set of criteria (§1.2.5) against which to measure the expressive capability and usefulness of belief revision systems;
- it introduces the *foundations approach* to belief revision (defined in §1.2.3 and justified in §1.2.3.3) that is to form the heart of IDNs.

Section 1.3 expounds the idea that inference need not be based on general inference rules that are based on implicit links between propositions (through pattern matching and formula manipulation), and that there are gains to be made in the explicit representation of particular inferences (as explicit links).

Finally, section 1.4 briefly introduces the notion of networks as an intuitively appealing yet powerful medium of expression. As such, networks provide the ability to marry the explicit representation of inferences as links with a foundations approach to belief revision in a principled manner that will support a formal interpretation structure (Chapter 3), a detailed analysis of algorithms (Chapter 4), and particular formulations of existing AI representations or problems (Chapter 5).

1.2 Existing Approaches to Belief Revision

1.2.1 Non-Monotonic Logics

In classical logics, be they propositional or first order, the main focus is on theorem-hood - does a set of axioms entail a particular proposition or formula? As knowledge representation languages, classical logics score highly in having a well-defined semantics. We know what it

means to say a particular statement is true or false, and given any statement we can interpret what it means, i.e. imagine what would have to be the case for it to be true or false. Classical logics also have the extremely useful property of being *monotonic*. As the amount of information in a theory or knowledge base Σ grows, the information derivable from Σ, its logical consequences, grows. Thus a definite answer to a query (i.e. π is a theorem, or $\neg\pi$ is a theorem) is guaranteed to remain the same no matter how Σ is extended[1].

So, given Σ encodes information about the animal kingdom or more specifically animals (denoted by the predicate A) all of whom are birds (denoted by B), mammals (M) or reptiles (R) and may or may not fly (F), we can ask questions about whether Polly the parrot (p), Larry the lizard (l) and Rocky the racoon (r) fly.

$$\begin{aligned}\Sigma = \{\ & \forall x.(A(x) \leftrightarrow B(x) \vee M(x) \vee R(x)),\\ & \forall x.(B(x) \rightarrow F(x)),\\ & \forall x.(R(x) \rightarrow \neg F(x))\ \}\end{aligned}$$

There are two possible approaches to ascertaining whether Polly being a bird, in conjunction with Σ entails that Polly can fly. The first is to consider all possible logical models that satisfy $\Sigma \cup \{\ B(p)\ \}$ and to see if F(p) is satisfied by them all, i.e. is true in all of them, in which case we write $\Sigma \cup \{\ B(p)\ \} \models F(p)$. Alternatively a syntactic *proof system* (e.g. natural deduction [Lemmon 1965] resolution [Robinson 1965] analytic or semantic tableau [Smullyan 1968] - alternatively see [Fitting 1990] for a general view) for manipulating formulae is used in an attempt to derive F(p) from $\Sigma \cup \{\ B(p)\ \}$, written $\Sigma \cup \{\ B(p)\ \} \vdash F(p)$. The essential property of such systems is that the transformation rules that are used are known to be *sound* or truth preserving (i.e. syntactic manipulation is incapable of producing false formulae from true premises or axioms) and *complete* (i.e. all true consequences of a set of axioms can be produced by syntactic manipulations).

Using natural deduction rules for the instantiation of universal quantifiers and the elimination of implications we can show that Polly can indeed fly

[1] Even if Σ becomes inconsistent, if a proposition π or $\neg\pi$ was previously a theorem, then it will remain a theorem. Unfortunately its negation will have also become a theorem.

$$\Sigma \cup \{ B(p) \} \vdash F(p)$$

but that Larry can't

$$\Sigma \cup \{ R(l) \} \vdash \neg F(l)$$

and that it is not possible to know if Rocky can or cannot fly but that either answer is consistent with the theory Σ

$$\Sigma \cup \{ M(r) \} \not\vdash F(r) \text{ and}$$
$$\Sigma \cup \{ M(r) \} \not\vdash \neg F(r).$$

The truth of F(p) or $\neg$F(l) is not subject to change given further additions to $\Sigma \cup \{ B(p), R(l), M(r) \}$ even if Larry were found out to be a flying iguana that is part bird.

Unfortunately logic is in some ways too much of a precise language. To say that every bird flies is to say exactly that – every bird flies **without** exception. Although in theory it is possible to use classical logics to write down exactly those conditions that make it possible for an individual to fly, as a modelling problem it seems an impossible task to keep track of the fact that birds can fly, unless they are penguins, unless of course they are penguins inside a plane, unless of course the plane is broken in which case they can't, unless of course the broken plane is being towed as a glider or is mounted on top of another plane ... The problem of writing down all the possible qualifications and exceptions in a finite way (it is always possible to do this by writing down all the facts as they become known but this will not be finite, given an infinite set of objects) is known as the *Qualification Problem* [McCarthy 1980].

Non-monotonic logics provide a way of bypassing or fudging this issue by allowing an increase in explicit information (Δ) to produce a decrease in implicit or derivable information. Thus it may be the case that $\Sigma \models \pi$ but that $\Sigma \cup \Delta \not\models \pi$. Thus without explicitly listing all the cases and possible exceptions to non-flying birds, it is possible to write a formal sentence that captures the intuition that birds normally fly.

1.2.2 Different Approaches to Non-Monotonic Logic

The basic principle behind all non-monotonic logics is some minimisation process. That is, given a set of axioms a closure is defined that adds some minimal amount of information, or somehow encodes the fact that what is written down is all that is known.

The simplest approach is that of *Closed World Databases* [Reiter 1978] or reasoning under the *Closed World Assumption (CWA)*. Given a theory[2] Γ the closure of Γ is given by the function[3] $Thms(\Gamma)$ where $Thms{:}Pw(\mathcal{L})\longrightarrow Pw(\mathcal{L})$ returns all the theorems derivable (in some language $\mathcal{L}$) given some proof system denoted by $\vdash$. The Closed World Assumption says that anything that is not provable from the original theory is assumed to be false. So, in the same way that a database containing all the relevant information about the world should return **all** aeroplanes that fly between Stansted and Edinburgh in response to question1 = "What airlines fly between Stansted and Edinburgh?", the database should answer "No" to question2 = "Does Aeroflot fly between Stansted and Edinburgh?" iff Aeroflot is not in the answer to question1. Applying this principle to Γ we find that the closure of Γ under the Closed World Assumption $CWA(\Gamma)$ is the closure of the original theory Γ extended by the negation of all the non-theorems of Γ:

(CWA) $\quad CWA = Thms(\Gamma\cup\Gamma_\Delta)$ where $\Gamma_\Delta = \{\ \neg\gamma \mid \gamma \notin Thms(\Gamma)\ \}$.

In his paper, Reiter shows that complex queries[4] of databases using the CWA can be answered without reference to the closure $CWA(\Gamma)$ as long as $\Gamma\cup\Gamma_\Delta$ is consistent, and that if Γ is consistent and consists of Horn clauses[5] then $\Gamma\cup\Gamma_\Delta$ is consistent.

The CWA can give us non-monotonic reasoning in two ways: directly, when $\Gamma\vdash_{CWA}$[6]$\neg\gamma$ for $\gamma \in \Gamma_\Delta$ but $\Gamma\cup\{\gamma\}\not\vdash_{CWA}\neg\gamma$; and indirectly, when $\Gamma\vdash_{CWA}\gamma'$ and $\Gamma\cup\{\gamma\}\not\vdash_{CWA}\gamma'$ where γ' depends on $\neg\gamma$ for its derivation. But this formulation of non-

[2] In the original paper Γ is assumed to be a deductive database but the principle applies to any logical language with a syntactic entailment $\vdash$.

[3] I shall use the notation f: X $\longrightarrow$ Y to denote the function f from domain X to codomain Y

[4] I.e. queries where compound formulae are constructed from atomic queries using $\neg$, $\wedge$,$\vee$ but do not contain quantifiers, although this restriction can be circumvented by using a slightly modified query.

[5] i.e. disjunctions having at most one positive (i.e. non-negated) literal (sub-formula). The use of Horn clauses in Γ is interesting for it foreshadows problems to arise when looking at TMSs.

[6] $\Gamma\vdash_{CWA}\alpha$ is shorthand for $\alpha\in CWA(\Gamma)$ or $\Gamma\cup\Gamma_\Delta\vdash\alpha$

monotonic reasoning is too restrictive. Firstly, one may not want to include every formula in the minimisation process. For example the database may not have information on my exact seat number, so in $CWA(\Gamma)$ it won't be 1a, 1b, nor 46e and so on – I won't have a seat on the plane! Things get worse if it's known that my seat number is one of a set of alternatives ($\exists x.$(seat-number(x) $\wedge$ sitting(guy, flight177, x))) for then a contradiction will exist in $CWA(\Gamma)$.

One way around the problem is to apply the CWA to selected predicates only, but there is still no guarantee that the closure will be consistent. A more flexible approach to minimising the information contained in Γ is to explicitly add a formula that states that the information encoded in Γ by a particular predicate P is all the information for which P holds. So rather than saying that if $\Gamma \not\vdash P(\bar{\gamma})$ then $\neg P(\bar{\gamma}) \in CWA(\Gamma)$ (where P is an n place predicate and $\bar{\gamma} = \gamma_1, \ldots, \gamma_n$) we get if $\Gamma \vdash P(\bar{\gamma}_i)$ for $i \in [1, m]$ then $\forall x.P(x) \leftrightarrow x = \bar{\gamma}_1 \vee \ldots \vee x = \bar{\gamma}_m$. This approach was developed by McCarthy [1980] and is called *Circumscription*. Formally, the circumscription of P in Γ ($CIRC(\Gamma, P)$) is given by the (second order) formula

(CIRC) $\quad CIRC(\Gamma, P) = \forall P'.[\, \Gamma(P') \wedge \forall \bar{x}.(P'(\bar{x}) \rightarrow P(\bar{x})) \rightarrow \forall \bar{x}.(P(\bar{x}) \rightarrow P'(\bar{x}))]$

where $\Gamma(P')$ is Γ with every occurrence of P replaced with P′. So the formulae reads, for every predicate P′ that satisfies Γ, if P′ satisfies all those formulae satisfied by P then P′ = P. The advantages of circumscription are that it necessitates adding just a single formula to Γ and that the minimisation is easily controlled. The disadvantages of using it are that it involves a second order formula quantified over predicates (although in many cases this can be reduced to first order[7]), and that in order to get any useful results (e.g. my database says the only flights to Edinburgh from Stansted are by Aeroflot, so if you want to fly from Stansted you must fly Aeroflot), the predicate P′ must first be guessed. This can be done by intuition or inspection on small examples but this is not a satisfactory long-term approach.

The Closed World Assumption and Circumscription both work by the minimisation of known information[8]. In the first case propositions not known to be true are assumed to be

[7] If Γ can be written as $\Delta(P) \wedge \forall \bar{x}.E(\bar{x}) \rightarrow P(\bar{x})$ where $\Delta(P)$ contains no positive occurrence of P and E is an expression containing no occurrence of P at all, then $CIRC(\Gamma, P) = \Delta(E) \wedge (E = P)$ where $\Delta(E)$ is $\Delta(P)$ with every occurrence of P replaced with E.

false, while in the second certain predicates are assumed to hold only for those facts for which they are known to hold. In addition to their individual problems, both these approaches suffer from a lack of expressiveness. It is not even possible to explicitly write a formula with the content "if π is unknown then assume π" (or alternatively "assume $\neg\pi$") or explicitly reason with it. This kind of reasoning must be done by the higher level functions of closing or circumscribing a theory making it impossible to reason about the absence of information within a particular theory (this being something we would want to do if we are attempting to capture expert knowledge).

A different approach, not involving minimisation, is taken by McDermott and Doyle [1980] in their Non-Monotonic Logic I (NML-I) where they augment first-order logic[9] with a modal operator **M** which has an intuitive reading of "it is consistent that ..." and can be prefixed to formulae π to give $\mathbf{M}\pi$, read as "it is consistent that π". This provides an explicit way of dealing with incomplete information and for making defaults explicit. In order to give an (operational) semantics for **M**, McDermott and Doyle define the non-monotonic theorems of a theory Γ ($NM'(\Gamma)$) by the recursive equation:

(NM′) $\qquad NM'(\Gamma) = Thms(\Gamma \cup \Gamma_\Delta)$

where $\Gamma_\Delta = \{\ \mathbf{M}\gamma \mid \neg\gamma \notin NM'(\Gamma) \wedge \mathrm{M}\gamma \notin Thms(\Gamma)\ \}$. This definition has a marked similarity with (CWA). If γ can't be proved then $\neg\gamma$ is added to Γ_Δ by (CWA) while $\mathrm{M}\neg\gamma$ is added by (NM′). A more important difference is that (NM′) is a recursive definition and there may be one, none or many fixed point solutions to (NM′), written $FP(\Gamma)$. How does one select the "real" theorems from all the possible answers? McDermott and Doyle would prefer to define the theorems of Γ under non-monotonic inference as the formulae in the smallest fixed point of NM', but this leads to obvious difficulties if there are no solutions or more than one minimal fixed point. A conservative approach is taken so that the "real" theorems non-monotonically derivable from Γ, $NM(\Gamma)$, are defined to be the intersection of all the fixed

[8] It is interesting to note that Shoham [1988] also uses minimisation as a way of doing non-monotonic reasoning but his work is based on the meta level construct of *preferences*. Rather than trying to change particular models so that they capture non-monotonic reasoning, his logic works on the idea of using only some preferred subset of the possible sets of models.

[9] In practice most of their work just involves simple propositions.

points, or the whole language if there are no fixed points:

(NM) $NM(\Gamma) = \bigcap_{\Psi \in FP(\Gamma)} \Psi$ if $FP(\Gamma) \neq \varnothing$

$= \mathcal{L}$ otherwise

In order to make the default inference that all Stansted to Edinburgh flights are by Aeroflot (unless known to be otherwise), we say

$$\forall f.\text{flight}(f) \wedge \text{connects}(f,\text{stansted},\text{edinburgh}) \wedge \mathbf{M}\text{carrier}(f,\text{aeroflot}) \rightarrow \text{carrier}(f,\text{aeroflot})$$[10].

Not surprisingly (given the difficulty of trying to characterise non-monotonic or default inferences) NML-1 has its problems. Unfortunately the major difficulty comes at the heart of the logic, in the intuitive reading of $M\pi$ as "it is consistent that π". There is no link between the truth value of $M\pi$ and $\neg\pi$ so both can appear in the same fixed point without causing a contradiction. In practice this only occurs when $M\pi$ occurs positively in a clause (i.e. is either an axiom or the left hand side of an implication) which should not happen given "correct" selection of Γ. However this is a very dangerous assumption to make, relying on a user of a logic to make sure it has the correct semantics.

Indeed, Moore [1983] defines a new logic *Autoepistemic Logic* by reconstructing the semantics of NML. This is done by interpreting $M\pi$ as $\neg L\neg\pi$ with $L\psi$ read as "it is known/believed that ψ" and extensions of Γ are solutions to

(AUTO′) $AUTO'(\Gamma) = Thms(\Gamma \cup \Gamma_\Delta)$

where $\Gamma_\Delta = \{ L\gamma \mid \gamma \in AUTO'(\Gamma) \} \cup \{ \neg L\gamma \mid \gamma \notin AUTO'(\Gamma) \}$. Under this definition, $M\pi$ (i.e. $\neg L\neg\pi$) and $\neg\pi$ generate a contradiction.

From a computational point of view, as first order logic (FOL) is semi-decidable[11] it is impossible to prove $M\pi$ true[12] in finite time for all π (because that requires $\Gamma \not\vdash p$) and given FOL's semi-decidability NML-1 is non-semidecidable. Given a particular formula π there is

[10] with appropriate additional axioms saying $\forall x,y.\ \text{carrier}(f, x) \wedge \text{carrier}(f, y) \rightarrow x = y$

[11] I.e. there are proof procedures which when asked if π is a theorem are guaranteed to terminate in finite time with an answer "yes" but may not terminate if the answer is "no".

[12] In either NML-1 or Autoepistemic Logic

no guarantee that any answer will be received in finite time about the truth or falsity of π.

A similar approach to default reasoning is taken by Reiter [1980]. Rather than including default rules as implications using M, default rules are explicitly coded as extra rules of inferences Δ attached to Γ. The inference rules are of the form

$$\alpha_1, ..., \alpha_n : \beta_1, ..., \beta_m / \gamma$$

and are intuitively read as "if $\alpha_1, ..., \alpha_n$ are true and it is consistent to believe (assume) $\beta_1, ..., \beta_m$ then γ should be believed (true/derived)". Yet again the same approach is taken to define the closure(s) or extension(s) of $<\Gamma,\Delta>$ (a theory now being a pair) as the minimal fixed point solutions to

(DL′) $$DL'(<\Gamma, \Delta>) = Thms(\Gamma \cup \Gamma_\Delta)$$

where

$$\Gamma_\Delta = \{ \gamma \mid \alpha_1, ..., \alpha_n : \beta_1, ..., \beta_m / \gamma \in \Delta \wedge \forall i.\alpha_i \in DL'(<\Gamma, \Delta>) \wedge \forall j.\neg\beta_j \notin DL'(<\Gamma, \Delta>) \}.$$

It might appear that this approach is less expressive than NML-1 or Autoepistemic Logic as the defaults are specified outside Γ and it is not possible to reason about default rules. However, this is not the case: Konolige [1987] shows that Autoepistemic Logic and Default Logic are equivalent.

One important result that holds for Default Logic [Reiter 1980] is that the extensions of $<\Gamma, \Delta>$ can be constructed in an iterative fashion:

$$E_0 = \Gamma;$$

$$E_{i+1} = Thms(E_i) \cup \{ \gamma \mid \alpha_1, ..., \alpha_n : \beta_1, ..., \beta_m / \gamma \in \Delta \wedge \forall i.\alpha_i \in E_i \wedge \forall j.\neg\beta_j \notin E \};$$

$$E \in DL(<\Gamma, \Delta>) \text{ iff } E = \bigcup_{i=0} E_i.$$

So starting with Γ and applying at each iteration a default rule whose antecedents are currently satisfied, if a point is reached when no more rules can be applied and nothing has been derived to invalidate all the rules that have been used, then the resulting set of theorems forms an extension of $<\Gamma, \Delta>$. By varying the order of application, different extensions may

be produced. By not taking the theorems of $<\Gamma, \Delta>$ to be the intersection of the fixed points but their union, Default Logic is radical (rather than conservative) and leads to cases where both π and $\neg\pi$ are theorems (but in different extensions).

Several interesting points can be highlighted by the study of non-monotonic logics. Most of the systems are computationally expensive, being defined in terms of closures of infinite sets involving non-semidecidable procedures or using second order logic. Many have problems with multiple extensions. The most important point is that although these systems to a certain extent can model changes in beliefs (i.e. propositions that are theorems under one theory may not be theorems in another), this process is cumbersome and not sufficient (on its own) for modelling changes in a theory.

1.2.3 Modelling the Dynamics of Epistemic States[13]

Unlike non-monotonic logics which are concerned with the retraction of certain designated inferences given the addition of contradictory information, Gardenfors' [1988] work is concerned with modelling the process of adding and subtracting information from *belief sets* through the processes of *expansion*, *contraction* and *revision*. The aim of his work is to present a series of postulates about what (epistemic) states should be considered as a suitable basis for building a theory of cognitive states (be they theoretic or realisable entities) and transitions between states. The definitions of postulates is motivated by several guiding principles and by intuitions of how belief sets should behave.

Presupposing a language containing propositions recursively defined using the standard propositional connectives ($\wedge$, $\vee$, $\neg$ and $\rightarrow$[14]), and some notion of logical consequence (i.e. provability: $\vdash$), contradiction and of the acceptance or rejection of propositions, the following *rationality* criteria for belief sets are advanced:

---

[13] The subtitle of "Knowledge in Flux" [Gardenfors 1988]

[14] note $\rightarrow$ is material implication

- "The set of accepted sentences should be consistent."
- "Logical consequences of what is accepted should also be accepted."

These criteria result in belief sets being defined as sets of formulae Γ of the language $\mathcal{L}$ s.t.

$$\Gamma = \{ \alpha \mid \Gamma \vdash \alpha \}$$

otherwise written as $\Gamma = Thms(\Gamma)$ where $Thms{:}Pw(\mathcal{L}) \longrightarrow Pw(\mathcal{L})$ returns all the theorems derivable in $\mathcal{L}$ given some proof system denoted by $\vdash$. The only requirements on $\vdash$ are that:

- if α is a tautology then $\vdash \alpha$;
- if $\vdash \alpha$ and $\vdash \alpha \rightarrow \beta$ then $\vdash \beta$ (i.e. Modus Ponens operates);
- $\alpha \vdash \beta$ iff $\vdash \alpha \rightarrow \beta$ (i.e. the deduction theorem holds)
- and that $\not\vdash \bot$,

where $\bot$ is a distinguished element representing contradiction.

These conditions on $\vdash$ ensure that $\vdash$ contains classical propositional logic. It is therefore possible to argue for or against Gardenfors postulates (as they stand) on the basis of properties of classical propositional logic. Gardenfors [1988, p24] does say that "if more is known or assumed about the structure of the object language, it may be possible to formulate other, more specific rationality criteria".

On this definition of belief sets, postulates are given that define: the expansion of a belief set K by the addition of α, written K^+_α; the contraction of K by α, K^-_α; and the revision of K by α, K^*_α. By assuming the Levi identity [Gardenfors 1988, p69] which states that $K^*_\alpha = (K^-_{\neg\alpha})^+_\alpha$, we need only to concern ourselves with the postulates for expansions and contractions. These basic postulates for expansions are as follows [Gardenfors 1988, p49-51]:

(K^+1) K^+_α is a belief set;

(K^+2) $\alpha \in K^+_\alpha$;

(K^+3) $K \subseteq K^+_\alpha$;

(K^+4) If $\alpha \in K$ then $K^+_\alpha = K$;

(K^+5) If $K \subseteq H$ then $K_\alpha^+ \subseteq H_\alpha^+$; and

(K^+6) for all belief sets K and all sentences α, K_α^+ is the smallest belief set that satisfies (K^+1) - (K^+6).

For contractions a similar set of basic postulates are expounded thus [Gardenfors 1988, p61-63]:

(K^-1) K_α^- is a belief set;

(K^-2) $K_\alpha^- \subseteq K$;

(K^-3) if $\alpha \notin K$ then $K_\alpha^- = K$;

(K^-4) if $\vdash \alpha$ then $\alpha \notin K_\alpha^-$

(K^-5) if $\alpha \in K$ then $K \subseteq (K_\alpha^-)_\alpha^+$;

(K^-6) if $\vdash \alpha \leftrightarrow \beta$ then $K_\alpha^- = K_\beta^-$

From these postulates a number of important consequences arise, in particular:

(3.21) If $\alpha \in K$ then $(K_\alpha^-)_\alpha^+ \subseteq K$ [Gardenfors 1988, p62]; and

Thm 3.1 The expansion function + satisfies (K^+1) - (K^+6) iff

$K_\alpha^+ = Thms(K \cup \{\alpha\})$ [Gardenfors 1988, p51].

An integral part of Gardenfors work is the view that belief sets are flat, i.e. they have no internal structure but are merely a set of propositions. Contraction and revision functions rely on maintaining consistent belief sets according to the principle of *informational economy*, i.e.

"When we change our beliefs we want to retain as much as possible of our old beliefs"

How beliefs have been derived is considered as irrelevant to the process of contracting or revising beliefs and is only considered when looking at "extra-logical information" that may help in making choices during the revision process. In support of his use of consistency as a guiding principle, Gardenfors quotes Harman [1986] who draws a distinction between *foundations theories* of revision on the one hand and *coherence theories* on the other. The latter are characterised by their use of consistency as a guiding principle, while the former depend on the structure of beliefs and their derivations.

Although agreeing with the goals and direction of Gardenfors work, I have a series of objections that prevent it forming a major part of this thesis. In brief his work is monotonic in nature; it relies on infinite closures of sets of axioms under entailment; it favours a coherence theory of revision over a foundations approach; and finally as it is based on, or capable of replicating propositional logic, any proposition is entailed by a contradiction. In the following section I shall examine these criticisms in depth and show how they support a foundations approach to belief revision.

1.2.3.1 Objections to Epistemic Dynamics

One of the goals of this research is to look at belief revision in general and one interesting application area is in non-monotonic style reasoning, a common everyday type of reasoning, as shown in §1.2.1. By placing a monotonicity constraint on expansions, i.e. $(K^+3) \equiv K \subseteq K^+_\alpha$, systems based on Gardenfors' postulates are forced to consider only monotonic systems of entailment. To see this, suppose $\vdash$ were non-monotonic and for some Γ,

$$\Gamma \vdash \beta \text{ but } \Gamma \cup \{\alpha\} \not\vdash \beta.$$

By (K^+3) if there existed any K s.t.

$$K \not\vdash \Gamma,\ K \not\vdash \alpha \text{ and } K \not\vdash \beta$$

then

$$K \subseteq K^+_\alpha$$

However

$$K^+_\Gamma \not\subseteq (K^+_\alpha)^+_\Gamma$$

as $\beta \in \mathit{Thms}(K \cup \Gamma)$ but $\beta \notin \mathit{Thms}(K \cup \{\alpha\} \cup \Gamma)$. This contradicts (K^+5). So either (K^+3), or (K^+5) must be given up, or $\vdash$ must be monotonic.

Another drawback to Gardenfors approach to reasoning is its reliance, as with many non-monotonic logics, on a fixed point construction over entailment. In the same way that this leads to computational intractability in non-monotonic logics, the need to construct

infinite sets of theorems leads to admitted computational problems [Gardenfors 1988, p36] in attempting to construct belief sets[15].

In order to demonstrate why I believe a foundations approach to revision is superior to a consistency approach I wish to do three things: show that (K^-5) leads to problems with retracting information and should be dropped; argue that an alternative to (K^-5) should be introduced and that this leads naturally to a foundations approach; and finally criticise Harman's rejection of a foundations approach.

Given (K^-5) and (3.21), if $\alpha \in K$ then $K = (K_\alpha^-)_\alpha^+$. Informally this means that in any state K the effect of retracting any information α can be counter-acted or nullified by the re-introduction of that information. I shall call this property *negative recoverability* to indicate the fact that is possible to recover from the removal of information by reasserting it.

This property has some undesirable consequences. Consider the proposition α and the minimal subsets Γ_i contained in K s.t. $\Gamma_i \vdash \alpha$. Furthermore, consider an operator $\otimes$ where

$$A \otimes B = \{ \{a, b\} \mid a \in A, b \in B \}$$

i.e. $A \otimes B$ is identical to $A \times B$ except the latter produces a set of ordered pairs or tuples whereas the former produces a set of sets.

As $\alpha \notin K_\alpha^-$ and K_α^- is closed under $\vdash$ then there must be some

$$\Gamma_\Delta \in$$
$$\Gamma_1 \otimes \ldots \otimes \Gamma_n$$
$$\text{s.t. } K_\alpha^- \cap \Gamma_\Delta = \varnothing.$$

I.e. at least one element in each Γ_i must have been removed from Γ to prevent the proof of α. Given this definition,

$$\forall \gamma \in \Gamma_\Delta.\ \gamma \notin K_\alpha^-.$$

But as $\gamma_i \in K$, and $K = (K_\alpha^-)_\alpha^+ = \mathit{Thms}(K_\alpha^- \cup \{\alpha\})$, then $K_\alpha^-, \alpha \vdash \gamma_i$. So[16], $K_\alpha^- \vdash \alpha \rightarrow \gamma_i$. As

[15] This particular problem has been resolved by Nebel - see § 1.2.3.4.

[16] by propositional rules of attachment and detachment - remember $\vdash$ is assumed to include propositional calculus.

K_α^- is closed under $\vdash$ (being a belief state) then $\alpha \rightarrow \gamma_i \in K_\alpha^-$. In particular, if $\forall i.\Gamma_i = \{\gamma_i\}$, i.e. α has a set of singleton "causes" $\gamma_i \rightarrow \alpha$ then (K^-5) forces us to include $\alpha \rightarrow \gamma_i$ in our belief state and a proposition becomes identified with its antecedents, $\alpha \leftrightarrow \gamma_i$.[17]

The result of all this is that when contracting a belief set, all the information required to restore the beliefs removed in the contraction must actually be included in that retraction! If I wish to contract K so α is removed, and in order to do this I must remove β, then I must also put $\alpha \rightarrow \beta$ in the contraction - this is so that when α is reasserted, β can be recovered.

It should be noted that a lot of this argument rests on well known problems that arise from the use of material implication to model causality and it might therefore be unfair to single out Gardenfors' work. However, he does choose to use material implication, and even if he did not use material implication similar problems remain. Given the definition of K^-5, if β depends on α and α is retracted, there must be some coding of this relationship in the resulting belief state so that when α is reasserted, β can be recovered.

As an example of the problems that occur in trying to retract information, assuming the language is propositional calculus using standard semantics and entailment, let $K = \mathit{Thms}\{ B \rightarrow F, P \rightarrow F \}$[18]. By asserting B then B, F, $F \rightarrow B$, $P \rightarrow B$, $\in K_B^+$. By asserting P then P, $F \rightarrow P$, $B \rightarrow P \in (K_B^+)_P^+$. If B is to be successfully retracted, either one of the proposition F or the proposition $F \rightarrow B$ and one of P or likewise $P \rightarrow B$ should be retracted. Similarly, if F is to be retracted, one of B or the implication $B \rightarrow F$ and one of P or (as before) $P \rightarrow F$ should be retracted. In the light of our original set of theorems, in the first case (retracting B) it would be preferable to reject the spurious implications $F \rightarrow B$ along with $P \rightarrow B$ as they were introduced to satisfy (K^-5). In the second case (retracting F) the implications form part of the original theory and it is more in accord with our intuitions to reject the atomic facts.

---

[17] In first order logic this also arises as a result of considering $K = \mathit{Thms}(K)$ for if $\alpha, \beta \in K$ then by strengthening the antecedent: $\gamma \rightarrow \alpha, \gamma \rightarrow \beta \in K$ for any γ and in particular $\alpha \leftrightarrow \beta$.

[18] One possible interpretation might be that B = "Tweety is a bird", P = "Tweety is in a plane" and F = "Tweety can fly"

In general, $\forall\alpha,\beta\in K.\ \alpha \rightarrow \beta \in K$ and one would expect the retraction of β to lead to: the removal of $\alpha \rightarrow \beta$ for all those facts α not used in the derivation of β; and the removal of α for the facts used to derive β. Thus determining the effect of retracting β requires information on how β was derived - this is surely a foundations theory of revision!

As a final remark, if the retraction of F leads to the removal of B and P, then asserting F leads to their reintroduction even when this may be undesired. For example, B and P may be two known causes of F, but there may be other causes for F and the assertion of F may be an observational report capable of occurring independently of B and P. In this case (K^-5) is too conservative in tying possible effects with causes.

Gardenfors considers the dual question: should $K = (K_\alpha^+)_\alpha^-$, for all K and α. This is a property I shall call *positive recoverability* in contrast to negative recoverability. His answer is no, for if $\neg\alpha \in K$ and $\neg\alpha \in H$ with $H \neq K$ then $H_\alpha^+ = K_\alpha^+ = K_\perp$ [19] and given $-_\alpha$ (contraction with respect to α) is a well-defined function, $K = (K_\alpha^+)_\alpha^- = (H_\alpha^+)_\alpha^- = H$ contradicting $H \neq K$. In order to defeat this objection one has to prevent a contradiction leading to the derivation of all possible facts. By using a foundations approach based on *relevance logic* firstly the source of a contradiction, a set of propositions $\Gamma_\perp$, can be isolated, and secondly the appearance of contradictions does not permit the derivation of any formula but merely prevents the derivation of those that depend on $\Gamma_\perp$. Thus $H_\alpha^+ \neq K_\alpha^+$ for $\neg\alpha \in K$ and H, and rather than having a single absurd belief set there are many.

Another reason (Gardenfors does not mention this) is that if $K = (K_\alpha^+)_\alpha^- = (K_\alpha^-)_\alpha^+$ then $-_\alpha$ and $+_\alpha$ are inverse functions for any α and are therefore isomorphisms and for all K, H, α, $K_\alpha^+ = H_\alpha^+ \Rightarrow K = H$ and similarly $K_\alpha^- = H_\alpha^- \Rightarrow K = H$. Thus for any given state there is only one other state from which it can be derived with a given epistemic input (i.e. an argument to a contraction or expansion function). This means that a series of updates and a final state enables the reconstruction of an initial state. Unless belief revision is deterministic, this argument implies that it is only possible to have one or other of these dual properties: i.e. positive or negative recoverability. Having rejected (K^-5) we are free to assume positive

[19] $K_\perp$ is the *absurd* belief set containing a contradiction and all that it entails.

recoverability.

1.2.3.2 Foundations vs Coherence

Rejecting (K^-5) and accepting positive recoverability (hence abbreviated PR) is equivalent to choosing between a foundations and a coherence theory of belief revision, for if $K = (K^+_\alpha)^-_\alpha$ then any β that can be derived in K using α, given $\beta \notin K$, is added when α is added and is removed when α is removed, unless an independent derivation has since been added.

Harman [1986] raises a series of objections rejecting a foundations theory in favour of a coherence theory. Before accepting the former these objections must be defeated. The primary objection relies on arguments about belief perseverance in humans [Ross et al 1980] where either a proposition π remains believed even after the evidence that originally led to its acceptance has been removed (and no other evidence that might support it has been added), or after evidence has been added that contradicts or undermines π.

Imagine that Karen is a student studying history[20] and that her grades in the past have been good. However, an aptitude test shows that she has very little aptitude for the subject, and she therefore concludes, after some deliberation necessary for her to choose how to revise her beliefs, that the course must have been easy and that she is in fact no good at history. On discovering that the aptitude test was incorrect (the scores were for another student of a similar name and unfortunately Karen's have been lost) what does Karen believe about her ability in history? Harman claims that belief perseverance would mean that she still felt that the course was easy and she had no ability. Therefore a foundations theory cannot be correct as $K \neq (K^+_\alpha)^-_\alpha$ for α = "aptitude tests reports lack of ability in history".

I would not argue with the scenario but with the description of the processes involved. Karen does not merely add and subsequently retract α: there is a further belief revision on K in order to remove the contradiction about $K \vdash \beta \wedge \neg\beta$, for β = "good at history". Thus the revisions are modelled by $K' = ((K^+_\alpha)^*_{\neg\beta})^-_\alpha = (((K^+_\alpha)^-_\beta)^+_{\neg\beta})^-_\alpha$. Now one would not expect, a priori, $\beta \in K'$ having removed it from K, unless its removal were solely dictated by the

[20] This is a cut down version of Harman's example.

addition of α. In other words, one model of Karen's belief revision is that the addition of α forces a contradiction over β which in turn leads to the retraction of β. The removal of α removes the contradiction which, according to the foundation theory should undo the retraction of β which was predicated on the presence of α. This assumes that the retraction process **was** predicated, i.e. dependent, on α. Imagine such a contradiction occurring - how does Karen go about deciding whether to accept $\neg\beta$ or alternatively throw out α? It is possible that determining factors in this resolution process are independent of the existence of the contradiction, e.g. friends had reported finding the course easy (which she may have discounted because she found it hard), and would persist after the contradiction is removed. Thus belief persistence can be explained in the way contradictions are resolved.

Another argument cited against the foundations approach is the psychological preference for *positive* rather than *negative undermining*. The principle of positive undermining states that

> "One should stop believing π whenever one positively believes one's reasons for believing P are no good",

while the principle of negative undermining states that

> "One should stop believing π whenever one does not associate one's belief in π with an adequate justification (either intrinsic or extrinsic)".

The first principle is associated with a coherence theory while the second defines a foundations approach. People in belief perseverance experiments keep unsupported beliefs (act in accordance with a coherence theory) but can be made to give up those beliefs if they are made aware of the principle of positive undermining, i.e. that they have no good reasons and all their reasons are no good. This, Harman claims, is further proof of the psychological validity of the coherence theory. However, the two principles appear to disagree only when there are no reasons for belief, in which case negative undermining says π should no longer be believed while positive undermining says π should still be believed (until some reason arises for it to be discarded, e.g. inconsistency, inelegance, compactness etc), so it is only this case that differentiates the principles.

If it is assumed that people use a self-referential or introspective argument as justification for their beliefs, e.g. "I believe π because π is currently in my belief system" or more specifically using a historical argument, e.g. "I believe π because at some point in the past was introduced into my belief system[21]", then all the observed phenomena that are explained by a coherence theory can be explained by a foundations theory. Positive and negative updating agree when other justifications are present, and if only the self-supporting argument remains then π still supports itself. Only when there is evidence that it is not possible for π to continue being believed is this self-referential argument removed, thus leaving π with no support thus necessitating its removal. Furthermore, one can envisage that self-referential justifications do not apply to all propositions, particularly when their derivation is obvious. However, it is part of the process of entrenchment to replace explicit justifications with implicit self-referential justification. For example, if I am told (by a reliable source) that a seminar on proof theory is being given tomorrow then I will not believe it because I believe it, but because someone told me. In contrast, if I was told that Santiago is the capital of Chile (I believe it is!) in school or read it in the paper I have lost those justifications for believing it but now believe it because I believe it!

1.2.3.3 Why Foundations?

Having shown the objections to a foundations approach are ill-conceived, why should a foundations theory actually be preferred (taking Occam's razor into account - i.e. if we don't need justifications why add them)? In the first instance, in order to reason in the face of contradictions, something that non-monotonic logics were devised to do (i.e. retract contradictory default inferences), a reasoner needs to know the following:

- what causes the contradiction, i.e. which propositions are inconsistent;
- what information (the *culprits*) is used to derive (*supports*) the contradictory propositions; and

[21] This introduction of new beliefs only being possible when there is some evidence, justification or motivation for doing so.

- how to decide which culprits are to be removed so as to guarantee the removal of the contradiction.

All this information is exactly what is maintained by a foundations approach to belief revision. By maintaining the information necessary to implement such a theory, a reasoner can easily deal with contradictions.

Secondly, there are the arguments between (K^-5) and PR, two mutually inconsistent postulates, the first stating that

$$K = (K_\alpha^-)_\alpha^+$$

while the second states that

$$K = (K_\alpha^+)_\alpha^-.$$

Which of these is the better principle to be included as a postulate about belief revision? Is it that having mistakenly added α, all traces of it (i.e. everything that was capable of derivation using α) can be removed by retracting it, or is it that having mistakenly removed α (and all information capable of deriving it), the belief set can be restored to its previous state? Now (K^-5) supports a coherence theory as

$$\forall\alpha,\beta.(\alpha \in K \wedge \beta \in K \Rightarrow \alpha \rightarrow \beta \in K)$$

thus all the propositions in a given belief set are identified with each other and the revision of the set of beliefs can be considered a global process. In contrast, as outlined above, PR supports a foundations theory for whenever new information is derived from newly asserted premises, e.g. $\beta \notin K$ but $\beta \in K_\alpha^+$, it must be retracted when that premise is retracted.

Harman claims that while the first has more psychological validity, the second is more principled. So, discarding motives of correspondence with how humans reason, and given we can model a coherence theory within a foundations theory, I propose that a foundations theory has a greater utility. In order to defeat Harman's previous objection to having (PR), namely that if $\neg\alpha \in K$ and H with $H \neq K$ then $H_\alpha^+ = K_\alpha^+ = K_\perp$ so either $K \neq (K_\alpha^+)_\alpha^-$ or $H \neq (H_\alpha^+)_\alpha^-$, the principle of a contradiction entailing everything must be removed.

1.2.3.4 Belief Bases

Nebel [1989] proposes doing belief revision on *belief bases* - defined as finite sets of propositions that (unlike Gardenfors' belief sets) do not have to be deductively closed. This removes the problems of: knowing whether to retract original or derived propositions: of retracting antecedents of implications (cf p14); and the computational problems that arise from using infinite closures.

However, it is interesting to note that the contraction function defined for belief bases does not (initially) satisfy the recoverability postulate K^-5! This postulate is only satisfied in general by extending the definition of contractions so that the proposition $B \vee \neg x$ is added to the contraction of B w.r.t. x. This is effectively **explicitly** encoding the original belief set in the contraction[22]. Nebel's proposed contraction function has additional problems. Either it has to treat multiple possible contractions as disjunctions, e.g.

$$(a \wedge a \rightarrow b \wedge b)_b^- = (a) \vee (a \rightarrow b)$$

or a selection function must be defined that relies on an ordering for all the propositions in a belief base. This ordering may itself be different for different bases and therefore require revision itself as transitions are made from one belief base to another.

Another interesting problem that arises when using belief bases is that of extracting information from a particular belief base. If one wants to know if a proposition holds in a belief base a proof must be constructed for it, or some reasoning performed. If the belief base changes and we still need to know the status of that particular proposition we must reconstruct the proof, or at the very least, recheck our old proof. What is this but Truth Maintenance at the (symbol) level of dependencies between propositions! This throws up an interesting parallel with the ATMS.

The ATMS could be viewed as operating on belief bases of assumptions, and similarly requires a secondary interpretation phase to determine the status of the nodes in the network given a particular current context (or belief base) or set of assumptions. But at least in the ATMS this can be done with a simple subsumption test on a node's label. Without storing

[22] Considering $\rightarrow$ as material implication, $B \vee \neg x \equiv x \rightarrow B$.

any information at the symbol level, belief revision performed solely using belief bases would be a cumbersome tool.

Nebel wishes to proceed in the same manner as Gardenfors (by giving definitions of belief states and transition functions between states) and do so on an abstract "knowledge" level, i.e. without reference to the structure of propositions or (non-logical) dependencies between them. Yet it is interesting to note that he himself believes [Nebel 1989, p308] that

> "Choosing the 'right' form of the premises seems to be one of the central tasks before any kind of belief revision can be applied."

I believe that belief revision of the kind proposed by Nebel and Gardenfors, operating solely at the knowledge level to be too difficult to manage without the kind of information contained at the symbol level, in propositions and the dependencies between them. Without this, one will always be searching for the right form of premises so that belief revision performs in the desired way.

1.2.4 Handling Contradictions

One of the problems with classical logic is that because of the definitions of entailment and implications, a contradiction (i.e. π and $\neg\pi$ occur in the same theory, or both are true in a particular model or interpretation) implies or entails any formula. This is obvious from the (truth table) definition of $\rightarrow$ whereby $\alpha \rightarrow \beta$ is true if α is false and in particular $\alpha \wedge \neg\alpha \rightarrow \beta$, or by using natural deduction rules:

(1)	$\neg\alpha$	
(2)	$\neg\alpha \vee \beta$	(1) $\vee$-I
(3)	$\alpha \rightarrow \beta$	(2) by definition or proof [Lemmon 1965, Thm48 p58]

Similarly, given semantic entailment is defined by $\alpha \models \beta$ iff $\mathcal{M}(\alpha) \subseteq \mathcal{M}(\beta)$ (where $\mathcal{M}(\alpha)$ are the models satisfying α) and no model by definition makes a proposition both true and false, $\mathcal{M}(\alpha \wedge \neg\alpha) = \varnothing \subseteq \mathcal{M}(\beta)$ for any β. So, in the presence of contradictions [Belnap 1977]

> "If ... (a computer reasoner) is a classical two value logician it must give up altogether talking about anything to anyone, or equivalently it must say everything to everybody ... contradictions are never isolated infecting as they do the whole system."

Non-monotonic logics are not a great help. They do provide a way of making unsound default inferences and retracting them when they would otherwise lead to contradictions, e.g. $M\alpha \rightarrow \alpha \mid\!\sim \alpha$ but $M\alpha \rightarrow \alpha, \neg\alpha \not\mid\!\sim \alpha$. However, if α and $\neg\alpha$ are true by virtue of first order (monotonic entailment) then the same "infection" occurs. Similarly, belief revision based on first order logic will generate an absurd belief set from the addition of α to a belief set containing $\neg\alpha$, $\neg\alpha \in K \Rightarrow K_\alpha^+ = K_\perp$. In order to avoid this, K must be revised through the removal of $\neg\alpha$ and the addition of α: $\not\vdash\alpha \Rightarrow (K^-_{\neg\alpha})^+_\alpha \neq K_\perp$. However, in attempting to assert π this approach commits a system to trying to prove π and to invalidating all proofs that exist before actually adding π.

Belnap [1976, 1977] proposes a logic whereby the introduction of a contradiction does not "infect" a theory to make all formulae theorems. Imagine a computer knows of certain propositions and that these may be marked as true and/or false, or have no value associated with them. These values are either asserted by external agents or derived from formulae that already have values. The possible combinations of being told or not told true and/ or false leads to four "truth values"[23] *n* (neither told true nor told false) *f* (told just false) *t* (told just true) and *b* (both told true and told false). Given an assignment s: $\Pi \longrightarrow V$ from atomic propositions $\pi \in \Pi$ to values $V = \{n, f, t, b\}$, s is extended to all formulae in the usual recursive fashion with negation defined by

α	*n*	*f*	*t*	*b*
$\neg\alpha$	*n*	*t*	*f*	*b*

and $\wedge$ and $\vee$ respectively defined as the meet (lowest upper bound) and join (greatest lower bound) operations over the lattice **L4**

[23] Fox [1990] holds that as there are in reality only two truth values there can only be two *told* "truth values", even though four "information-states" exist. This insistence on only two external truth values means that even if nothing is known about π, it is either true or false and in any case $\pi \vee \neg\pi$ is true. This approach is similar to that of Van Fraassen's *supervaluations* [1966, 1968, 1969] in which formulae lacking truth values for some component are evaluated under all possible assignments and if all assignments agree then this value is used.

Using the definition of material implication ($\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$) we get

		β			
	$\alpha \rightarrow \beta$	*n*	*f*	*t*	*b*
α	*n*	*n*	*n*	*t*	*t*
	f	*t*	*t*	*t*	*t*
	t	*n*	*f*	*t*	*t*
	b	*t*	*b*	*t*	*b*

Although $s(\alpha \rightarrow \beta) = t$ if $s(\alpha) = f$, if $s(\alpha) = b$ (i.e. α is contradictory or inconsistent) then $s(\alpha \rightarrow \beta) \in \{b, t\}$ and it is impossible to assign a true value to the implication solely on the basis of a contradiction. Similarly from $s(\alpha \rightarrow \beta) = t$ and $s(\alpha) = b$ it is impossible to say what value β must have.

Entailment is defined by $\alpha|\!-\beta$ (Belnap actually writes $\alpha \rightarrow \beta$ but I will reserve $\rightarrow$ as a symbol within the language) iff $s(\alpha) \leq s(\beta)$ for all assignments *s*. Interestingly (although not totally unexpectedly, given this work is equivalent to Anderson and Belnap's relevance logic [1975]) if we construct a new implication based on this principle where

$s(\alpha \longmapsto \beta) = t$ iff $s(\alpha) \leq s(\beta)$,
$s(\alpha \longmapsto \beta) = f$ iff $s(\alpha) > s(\beta)$,
$s(\alpha \longmapsto \beta) = u$ otherwise

then it is impossible to construct a truth-functional characterisation of $\longmapsto$ using $\wedge$, $\vee$, and $\neg$.

Ginsberg [1988] generalises Belnap's work by defining a *bi-lattice* as two partial orders $\leq_k$, and $\leq_t$ over a set B with glbs $\vee$, + and lubs $\wedge$, · respectively, and a negation operation $\neg$: B $\longrightarrow$ B s.t.

(1) $(B, \wedge, \vee)$ and $(B, \cdot, +)$ are complete lattices

(i.e. $b_1 \oplus b_2 \in B$ for all $b_1, b_2 \in B$, $\oplus \in \{ \wedge, \vee, \cdot, + \}$.)

(2) $\forall b \in B.\ \neg(\neg b) = b$

(3) $\forall a,b \in B.\ \neg(a \wedge b) = (\neg a) \vee (\neg b)$ and $\forall a,b \in B.\ \neg(a \vee b) = (\neg a) \wedge (\neg b)$

(4) $\forall a,b \in B.\ \neg(a \cdot b) = (\neg a) \cdot (\neg b)$ and $\forall a,b \in B.\ \neg(a + b) = (\neg a) + (\neg b)$

Obviously **L4** fulfills these criteria and for any bi-lattice the elements of **L4** will appear as distinguished elements n', f', t', and b' s.t. $\forall v \in B.(n' \leq_k v \wedge v \leq_k f' \wedge t' \leq_t v \wedge v \leq_t b')$. Given an assignment ϕ: $\mathcal{L} \longrightarrow B$, if $\phi(\alpha) \leq_k \phi(\beta)$ then β contains at least as much information or knowledge as α and if $\phi(\alpha) \leq_t \phi(\beta)$ then β is at least as truthful as α.

Briefly inference is characterised as the process of *closing* a truth assignment ϕ by choosing those assignments ψ that contain just enough extra information, i.e. $\forall \alpha \in \mathcal{L}.\phi(\alpha) \leq_k \psi(\alpha)$ (if ψ fulfills this condition then ψ is an *extension* of ϕ) so that if $\alpha \models \beta$ then $\psi(\alpha) \leq_t \psi(\beta)$ (if ψ fulfills this condition then ψ is *apparently closed*).[24] Having found the apparently closed extensions of ϕ, $A(\phi)$, the closure could be defined as the pointwise k-minimum value of each formula[25]:

$cl(\phi)(\alpha) = \psi_1(\alpha) \cdot \ldots \cdot \psi_n(\alpha)$ for $\psi_i \in A(\phi)$.

Some apparently closed extensions may draw stronger conclusions than are warranted by the evidence but if $\psi_1 \leq_k \psi_2$ (i.e. $\forall \alpha \in \mathcal{L}.\psi_1(\alpha) \leq_k \psi_2(\alpha)$) then ψ_2 is filtered out by the minimisation. However it is possible that ψ_2 makes some (unjustified) inference that increases the value of one proposition while preventing an increase in another so that neither $\psi_1 \leq_k \psi_2$ nor $\psi_2 \leq_k \psi_1$. An additional order relation on assignments is defined with

$\phi <_I \psi \equiv \phi \leq_k \psi \vee \exists \pi.(\phi(\pi_1) + \ldots + \phi(\pi_n) \leq_k \psi(pi)$

where $\pi, \pi_i \in \Delta = \{ \alpha \mid \psi(\alpha) \neq \phi(\alpha) \}$

---

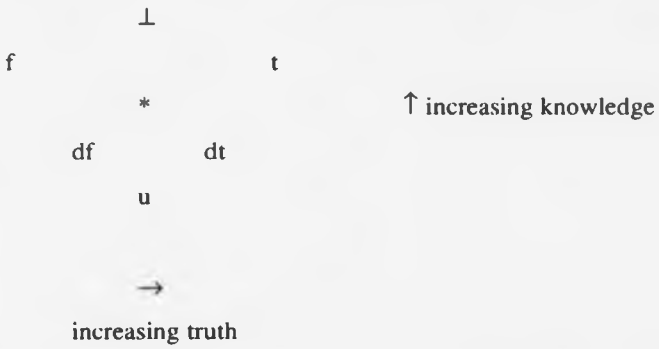
[24] For brevity I have skipped extra conditions that are necessary for ψ to be apparently closed. These fulfill the role of making sure that the information contained in each compound formula is as much as the information contained in the subformulae.

[25] This is analogous to taking the intersection of extensions in NML-1 to obtain the set of non-monotonic theorems, or to any other *cautious* approach to non-monotonic reasoning.

So the closure is defined as the meet over the <|-minimal apparently closed extensions of ϕ:

$$cl(\phi) = \Pi \{\psi \in A(\phi) \mid \psi' \in A(\phi), \psi' <| \psi \Rightarrow \psi' = \psi\}$$

In order to show how to perform default reasoning using lattices the lattice **D** is defined.



Imagine we have propositions Tweety is a bird (b), Tweety is a penguin (p), and Tweety flies (f), and we have a inference rules $b \rightarrow f$[26] and $p \rightarrow \neg f$. Now birds by default fly so the inference $b \rightarrow f$ is assigned a value dt while penguins not flying is a "real" or definite inference rule and is assigned the value *t*. Given an assignment

π	b	p	f	$b \rightarrow f$	$p \rightarrow \neg f$
$\phi(\pi)$	*t*	*u*	*u*	*dt*	*t*

and given $b \wedge b \rightarrow f \vdash f$, then $cl(\phi)(f) \geq_k dt \wedge dt = dt$. Continuing in this fashion the closure can be constructed:

π	b	p	f	$b \rightarrow f$	$p \rightarrow \neg f$
$cl(\phi)(\pi)$	*t*	*df*	*dt*	*dt*	*t*

If it is asserted that Tweety is in fact a penguin then the closure of the resulting extension ψ of ϕ is:

[26] The notation here is a little vague - $\alpha \rightarrow \beta$ is really an inference rule but it acts like an implication within the language and can be assigned an element in **D**.

π	b	p	f	$b \rightarrow f$	$p \rightarrow \neg f$
$cl(\phi)(\pi)$	*t*	*df*	*dt*	*dt*	*t*
$\psi(\pi)$	*t*	*t*	*dt*	*dt*	*t*
$cl(\psi)(\pi)$	*t*	*t*	*f*	*f*	*t*

So increasing the knowledge (about p) has led to a decrease in truth (about f) and the reasoning is non-monotonic. However this non-monotonicity in truth is bought at the expense of monotonicity of knowledge - it is impossible to throw away knowledge, be it asserted or derived, unless one goes back to a basic starting assignment for facts known about the world and then recloses it.

This approach to reasoning and representation is useful in that contradictions do not dominate theories so that anything is derivable from a contradiction. However, as with normal logics, the problems of updating and revising beliefs in the absence of explicit justifications remain. This is readily admitted by Ginsberg when he says that when using **D** one "must essentially re-derive all of our earlier conclusions at every step" [Ginberg 1988, p278]. He claims this is necessary for any default reasoning system, but I hope to show in this thesis that this is not the case.

1.2.5 Criteria for Belief Revision

In this section I have looked at various ways of capturing everyday default or non-monotonic reasoning. Any system that attempts to capture the process of doing belief revision must satisfy five main criteria:

(i) non-monotonicity

the system must be non-monotonic, capable of undoing inferences that have previously been made;

(ii) justifications

the system must keep track of justification information in order to allow the retraction of information in general and contradictions in particular;

(iii) contradictions

more strongly, the system must be able to handle contradictions by preventing their occurrence "infecting" the system allowing anything whatsoever to be derived;

(iv) reference to lack of information

the system must be able to explicitly refer to the lack of information about some proposition and make inferences of the form "if I don't know anything about π then infer ψ";

(v) computational tractability

and finally, to be useful a system should be computationally tractable, an issue that in many cases has been ignored by non-monotonic formalisms to date.

1.3 Capturing Consequences

The purpose of formalism is to capture information about, or construct a model (in the informal sense of the word) of some domain so that a compact representation can be used to derive information about that domain without interacting with the domain directly. In order to make sense of such formalisms, an interpretation of the elements of the formalism is needed. In the case of logic this is a mapping of logical symbols to objects (be they individuals, properties or relationships) along with a truth assignment that indicates whether the relationships between elements of the formalism holds in the domain. The formula

$$B(p) \wedge F(p)$$

could be interpreted as follows: p denotes Polly my parrot, B is the property of being a bird, and F is that of flying, and they are both deemed to be true of Polly.

A logical model of a set of sentences Γ is an interpretation in which all the sentences in Γ are true. So the process of capturing knowledge about some domain requires the construction of some set of sentences (an axiomatisation Γ) that represents some explicit information in such a way that any logical model of Γ (i.e. an interpretation in which all $\gamma \in \Gamma$ are true) is a logical model of the desired implicit information Θ that is to be captured. Semantically the axiomatisation Γ restricts the set of possible interpretations so that any model of Γ is a model of Θ. Any formula θ that is true in all models of Γ is a logical

consequence of Γ and is said to be entailed by Γ. The complete set of implicit information entailed by Γ is a logical theory (of Γ).

In §1.2 it was shown how non-monotonic logics were developed from particular definitions of entailment (or extensions or closures) so that simple "common sense" theories could be captured by particular, intuitive, axiomatisations. Such work proceeds by choosing (or having preconceived ideas about) pairs $<\Gamma,\Theta>$ of explicit and implicit information and attempting to define (through the definition of entailment) a function that captures this relationship. This can be difficult (as with the Yale Shooting Stick Problem [Hanks and McDermott 1986])!

Rather than working on the basis of Γ defining Θ in model theoretic terms and/or some associated proof theory using implicit links between propositions, I maintain that representations of domains should explicitly include the potential inferences that can be made, in terms of direct links between propositions, and that reasoning is a process of using those links to manipulate and/or create interpretations of those propositions.

1.3.1 Mental Models

Johnson-Laird [1983] proposes and defends the thesis that

> "reasoning ordinarily proceeds without recourse to a mental logic with formal rules of inference";

that is, rules that take a set of propositions (about the world or some abstract domain) and through purely syntactic manipulation return new propositions. The fundamental idea behind reasoning without rules of inference is that people construct *mental models* from a set of sentences (the antecedents or premises of a formal inference), symbolic representations of states of affairs, and manipulate these models to test the validity of relationships (conclusions or consequences) between objects or sets of objects in the model.

For example, having been told that all men are mortal and that Socrates is a man a model is constructed in which a representative set of men (this set is open to arbitrary extension at any time) are all designated as mortal and in which Socrates is identified as a particular case of a man. Therefore a link has been established between Socrates and those

individuals that are mortal and the conclusion that Socrates is mortal can be drawn.

		man =	mortal
Socrates	=	man =	mortal
		(man) =	mortal
			mortal
			(mortal)

Socrates is a man all men are mortal

[the brackets around "man" and "mortal" are to suggest that these objects and their relationships can be replicated indefinitely] If an inference is to be *implicit* (i.e. occur unconsciously as a natural side effect of comprehension of written or verbal communication) then the construction of a model that satisfies the premises is sufficient. If an inference is to be *explicit* then a model generated to satisfy the premises must be tested to see if it is possible to alter the model so that it still satisfies the antecedents but the conclusion no longer holds. In the case of Socrates it is not possible to add or remove objects and/or links so that Socrates is still a man, all the men in the model are shown to be mortal, but Socrates is not mortal.

The interesting point to come out of Johnson-Laird's work is not that he can explain and predict how humans will, in general, cope with reasoning tasks but his idea that one explicitly manipulates models rather than using logical statements to restrict the class of models, and then uses rules of inference to find out what holds in all of the models of this restricted class.

1.3.2 Mental Models in AI

When using logic to capture a body of knowledge it is sufficient to write down an axiomatisation without explicitly specifying the deductions that are to be made from those axioms. The rules of deduction for the logic will generate the set of consequences for that axiomatisation. Most AI systems not only capture a specific body of knowledge in terms of objects and the relationships between them but also explicitly say how that information is to be used to make specific inferences. Thus the actual information being encoded also specifies

how that information is to be used.

A prime example is that of production rule or rule-based systems where the rules themselves are a cross between a declaration of relationships between objects and real rules of inference. A rule of the form "$p(x) \rightarrow q(x)$" used in a forward chaining system not only says that property q holds for all things satisfying p, but also that having had p(a) derived or placed in the working memory for some a then derive q(a). Alternatively a prolog or backward chaining rule of the form "$p(X) :- q(X)$" says that in order to derive p(X) for some X, prove q(X). The "is-a" link "$p \longrightarrow q$" in a semantic network is again both a declarative statement (things of type p are also things of type q) and an inference rule (things that are p inherit characteristics of q's).

The same could not be said to apply to material implications of the form $\forall x.p(x) \rightarrow q(x)$[27]. This statement does express a relationship between the properties p and q but it gives no guide as to its use. It could be used to prove q(a) given p(a), or $\neg p(a)$ given $\neg q(a)$, or $\forall x.p(x) \rightarrow r(x)$ given $\forall x.q(x) \rightarrow r(x)$ or a host of other consequences involving $\forall x.p(x) \rightarrow q(x)$.

It should be noted that saying "rules specify how information is to be used" is somewhat of a simplification when meta-level reasoning systems are taken into consideration. There are systems (see [van Harmelen 1991] for a review) that provide a meta-level language for specifying how information at the object level should be used or interpreted. Most of these systems[28] are based on explicitly representing what has previously been implicit control information. This means a control structure can itself be reasoned about and modified in particular situations. This does not however alter the basic point that many AI representations encode how information is to be used. Adding meta-level reasoning normally extends this by giving the power to say **when** information is to be used.

---

[27] the nearest thing that logic gets to a rule of inference actually in the object language, given $\alpha \rightarrow \beta$ iff $\alpha \models \beta$

[28] This includes those systems whose focus of action is either - to use van Harmelen categorisation - in the object level or is mixed between the object- and meta-levels rather than those whose focus is primarily in the meta-level.

In as much as AI systems can be said to embody particular theories about the world or particular expert domains, these theories represent a model of the inferences about a particular domain. Manipulating the model corresponds to applying particular rules to reach a desired goal. Non-monotonic reasoning systems have difficulty in describing just exactly what inferences should and should not be made. In particular the existence of multiple extensions, for example the Yale shooting stick problem [Hanks and McDermott 1986], is problematic with much (naive) discussion as to what inferences are intuitive or one would expect. It may even be possible that the whole field of non-monotonic reasoning (or at least the intuitive modeling thereof) is doomed to failure by looking for general principles of reasoning where they may not exist. This supports the idea that part of the process of modelling a domain should be the elicitation of exactly how information is used to derive new information.

1.3.3 Causal Models

Intuitively, causality is a simple conceptual notion: α causes β if whenever α occurs then β occurs as a result of α, and β would not have occurred without the occurrence of α or some other cause of β. However, the concurrency of α and β is insufficient to distinguish between genuine causality, reverse causality (α really causes β but it is perceived that β causes α e.g. the idea that images in the eye are produced by the emission of some substance that illuminates the object rather than the reception of light from those objects), spurious causality resulting from statistical correlation (e.g. being red causes apples to be sweet), and epiphenomenalism (in which α is a "side" effect of some genuine cause γ of β, thus it appears that α causes β).

In the Situation Calculus [McCarthy and Hayes, 1969] causality is expressed as an (material) implication between pre-conditions of an action holding in a situation s and the result of that action holding in some future situation s'. However, given the paradoxes of material implication, in any situation that does not satisfy some pre-conditions Γ, a material implication $\Gamma \rightarrow \beta$ could represent a causal connection between Γ and β. So "if I am the king of France then I am bald" is a true implication and there is a causal connection between being the king of France and being bald. Ginsberg [1985] presents a way of analysing such

counter-factual implications (where the antecedent is false) in terms of *possible worlds*: α counter-factually implies β in world Γ iff in all worlds most similar to Γ in which α is true then β is true. Counter-factuals are loosely related to causality for many counter-factuals capture a causal relationship. For example given "the dinner will be on time if the power in the kitchen doesn't fail"[29], the power failure is seen as a cause for the lateness of dinner. Intuitively this is what it seems to mean for α to cause β: if α can be made to be true (by some action which moves us to a different world or situation) then one would expect β to subsequently happen. However, being a true counter-factual is insufficient evidence of a causal relationship. Although it is the case that "if the power does not fail then the guests will arrive on time" is a true counter-factual, moving to the nearest possible world (in which the power fails) does not affect the arrival of the guests and there is no causal connection.

Most AI systems that attempt to capture causality work with a built-in model of the causal connections. The causal links are explicitly modelled as such but there are no reasons for supposing that given a "causal connection" in the model that one actually exists. This applies not only to using implication as a causal link but to statistical causal models (e.g. [Pearl], Prospector) or semantic networks (e.g. CASNET) or to ad-hoc knowledge representations. What these systems do share is an explicit representation of causal links and a process of reasoning about the causal model by manipulating or passing information along such links. This means that

> "... the reasoner can succinctly axiomatise the domain-dependent information needed to infer that a ... (property) persists and reason about persistences in the same manner as changes" [Weber 1989]

1.4 Networks

The choice of a network representation is based on three factors: intuitive appeal; familiarity; and computational considerations.

---

[29] this example is from Ginsberg [1985]

The intuitive appeal of networks is founded on the apparent accessibility of graphical relationships. This is one of the justifications of such approaches as: semantic networks where inference can be performed through simple graph traversal algorithms, though the representation is no more powerful than restricted predicate calculus; or the use of graph reduction techniques to Rule Based Systems using certainty factors [van der Gaag 1987]. Another benefit is provided by the topological clustering of information, and although much of this benefit is available to linear representations given appropriate indexing schemes, graphical representations offer the efficiency and flexibility of special purpose reasoning algorithms.

The drawback of such graphical systems also lies in this flexibility that they offer: it is easy to define an algorithm and representation with no clear idea of what the algorithm does, nor what the representation means, other than pointing to the results in particular situations. For example this has led to myriad different interpretations for the "is-a" link in semantic networks (see §5.4).

One of the goals of this thesis is to provide a principled way of defining network-based belief revision systems so as to prevent such problems arising. In the case of different approaches to the same problem, (i.e. more than one way of interpreting the same type of network), the relevant definitions will be available in order to highlight the differences between interpretations.

The familiarity of networks as representations in AI, Computer Science and Mathematics has led to the development of a wide ranging vocabulary to describe properties of networks. Graph-related topological concepts such as connectedness, directedness and cyclicity all provide useful ways of classifying particular types of networks.

Such vocabulary occurs in diverse areas ranging from the structure of arguments [Loui 1987, Lin and Shoham 1989] to constraint satisfaction problems [Dechter and Pearl 1988]. In this thesis concepts such as directed graphs, cycles and acyclic networks, cutsets and reducible graphs will play a key role. Additionally, because of a large body of work a number of existing graph theory algorithms can be used or modified for use in the algorithms presented in Chapter 4.

Finally, the use of graph-based representations raises the hope of directly transferring the processing of network-based algorithms to parallel computation devices, using the topology of the knowledge representation network to determine the connectivity of processing elements and the passing of information between them. Although this is not investigated in depth in this thesis, passing attention is paid to existing examples of this idea. In particular semantic networks are particularly amenable to such highly parallel architectures [Bic 1985 and Fahlman 1980] and attempts have been made at applying this to Assumption Based Truth Maintenance Systems [Dixon 1988].

1.5 Conclusion

The main thrust of this chapter has been to examine why existing work in AI fails to provide an adequate approach to belief revision, that is, the management of changes in some explicit body of knowledge. In the course of this examination a set of criteria have been established which serve as a guide to the flexibility and utility of belief revision systems. It has also been proposed that a *foundations* approach to belief revision offers the best way of fulfilling those criteria.

This chapter has also put forward the idea that inference can be performed through the explicit representation of links between antecedent and consequent and the direct interrogation or manipulation of such links. Finally several reasons have been put forward for a preference for network-based algorithms, providing a clear semantics is given.

The following chapter focuses on *Truth Maintenance Systems* (TMSs) - a class of network-based systems capable of:

- supporting a foundations approach to belief revision [Gardenfors 1988, p35];
- explicitly representing rules of inference [Morris 1987]; and
- meeting all the belief revision system criteria established in §1.2.5.

Chapter 2 will contain:

- a review of the developments of TMSs and of the major developments contained in the various approaches taken by different systems;

- an examination of unifications of the different approaches;
- an examination of the qualitatively different approaches to such concepts as defaults, premises and assumptions;
- a survey of existing applications and how they are suited to different TMS approaches; and
- an examination of some problems that arise in trying to apply such systems.

This will in turn lead, in Chapter 3, to the development of a general framework for specifying and implementing belief revision systems.

CHAPTER 2

Truth Maintenance Systems

Truth Maintenance Systems are a class of systems that have been developed to act as intelligent caches of information and to provide certain book-keeping functions. In particular TMSs are used to record inferences and resolve inconsistencies that may arise and as such TMSs act as mechanisms for doing belief revision. Of the five characteristics that are desirable for belief revision systems (see §1.2.5), all TMSs provide mechanisms for resolving contradictions, and all use some form of dependency to record the sources of support for propositions with the majority using uni-directional justifications. The three remaining characteristics of non-monotonicity, computational tractability and ability to explicitly represent reasoning that involves the absence of information, are possessed in a variety of combinations by different systems.

This correspondence between TMSs and the belief revision criteria indicates that TMSs are prime candidates for building general belief revision systems. To this end, the three main styles of TMS will be examined in §2.1 with differences in style and application being discussed in §2.3. On a more formal level, various unifications have been proposed and these will be discussed in §2.2 while the applicability (or otherwise) of Gardenfors' to TMSs will be examined in §2.4. The classification of various TMS activities and the identification of common elements of structure and functionality possessed by TMSs that arise from this chapter will be used in Chapter 3 to define a general approach to Truth Maintenance and belief revision, the *Interpreted Dependency Network*, that forms the main contribution of this thesis.

2.1 Development of TMSs

Truth Maintenance Systems first emerged as individual differentiated mechanisms with Doyle's seminal paper [Doyle 1979] that represents the culmination of work started by Stallman and Sussman [1976] and developed further by de Kleer et al [1977]. Developments included a different style of Truth Maintenance by McAllester [1978, 1980] and a further change in style by Martins and Shapiro [1983] that was developed by de Kleer [1986].

The original idea behind TMSs is to act as a mechanism for providing efficient management of search. As a problem solver develops a line of reasoning the premises and

conclusion of arguments are recorded as propositions or *nodes* in the TMS, and the inference is represented by a *dependency* or *justification* linking the antecedents to the consequent. The validity of the lines of reasoning are indicated by the *labels* explicitly assigned to particular nodes and derived for all other nodes and justifications. The benefits of using a TMS come from the way contradictions or dead ends in the search space are handled. Having ascertained that a particular line of reasoning generates a contradiction (and has therefore ended in failure), some action is taken to invalidate parts of that argument, potentially invalidating arguments supporting other nodes or even giving rise to new arguments based on non-monotonic justifications. Because the TMS records failures, and more importantly the underlying **reasons** for those failures, the problem solver linked to the TMS is guaranteed never to explore lines of reasoning known to produce contradictions.

2.1.1 Justification-Based Truth Maintenance

The TMS presented by Doyle in [1979a] and [1979b] - henceforth referred to as a justification-based TMS or JTMS - is a very general system capable of representing premises, assumptions and contingent beliefs. The difference between premises, assumptions and contingent beliefs is not explicit, but comes from the structure of the dependencies: a premise will not be dependent on any other nodes; an assumption will be dependent on the disbelief of some node (e.g. "not A" not being believed provides support for believing "A" is true is equivalent to assuming "A" unless "not A" is believed); and a contingent belief is a node that has some dependencies but cannot be classed as an assumption.

The purpose of the JTMS is to provide a consistent interpretation of the nodes by assigning a value of "believed" or "not believed" to each node on the basis of whether that node has a valid justification, or reason for belief, making it *in* the current set of beliefs, or not, in which case the node will be *out* i.e. not *in*. Doyle identifies two main types of justification although it would be easy to define additional types[1] The first is the "support-list" (SL) justification of the form "If a_1 ,..., a_m are believed and b_1 ,..., b_p are not believed

[1] However, as the TMS is designed to be a domain independent system, the types of justification should represent general kinds of justification and not be domain specific.

then n is believed", written

$$(a_1, ..., a_m)_{in}\ (b_1, ..., b_p)_{out} \longrightarrow_{SL} n.$$

The second is the "conditional proof" (CP) justification of the form "If c is believed whenever $a_1, ..., a_m$ are believed and $b_1, ..., b_p$ are not believed then n is believed", written

$$(a_1, ..., a_m)_{in}\ (b_1, ..., b_p)_{out}\ c_{cons} \longrightarrow_{CP} n.$$

The a_i's are called "in-nodes or -hypotheses" (for SL- or CP-justifications respectively), the b_j's are "out-nodes or -hypotheses" and c is the "CP-consequent". Whenever a dependency d supports a value *in* we say that d has a value *in* or is *valid*. So, using this notation we could differentiate the following:

a premise P	justified by	$\longrightarrow_{SL} P$
an assumption A	justified by	$(\neg A)_{out} \longrightarrow_{SL} A$
	or more generally	$(a_1, ..., a_m)_{in}\ (b_1, ..., b_p)_{out} \longrightarrow_{SL} A$
a contingent belief C	justified with	$(a_1, ..., a_m)_{in} \longrightarrow_{SL} C$

where a_i and b_j are arbitrary nodes.

The basic interactions between the problem solver and the JTMS consist of the problem solver asserting and retracting justifications and marking nodes as contradictions[2] with the TMS answering queries about the status of particular nodes. The addition or removal of a justification from a node n invokes the "Truth Maintenance" (TM) procedure which updates n's value so that it corresponds with that supported by n's justifications. I.e. if all the justifications are *out* n should be *out* but if one or more justifications are *in* then n should be *in*. Any change in n's value is propagated to the *consequences* of n, i.e. those nodes whose values depend (in part) on n's value. On termination of TM no node's value should conflict with that supported by its justifications.

Doyle's [1979] algorithm works by constructing a list of nodes whose value is dependent on n (where n is the node where the justification is added or removed) and by

[2] There is no need to explicitly create nodes as this is done automatically when they are mentioned in some justification.

labelling them *nil*: successive attempts are made to determine a value for each node by considering only those nodes labelled *in* or *out*, and then by assuming that all *nil* values will become *out*. The assumption that *nil* values become *out* is a relaxation process designed to assign values to nodes in cycles whose values would otherwise be incalculable. This process is explicitly captured by Goodwin [1987] in an improved version of Doyle's algorithm which explicitly deals with strongly connected components[3].

When a node is marked as contradictory if that node is *in* then a Dependency Directed Backtracking (DDB) procedure is called (similarly if a previously marked contradictory node is *in* at the end of TM then DDB is also invoked). DDB traces back through the nodes supporting the contradictory node, say c, to discover the underlying inconsistent assumptions $S = \{A_1, \ldots, A_n\}$, i.e. those nodes that support c (either directly or through a chain of other derived nodes) and who are in turn supported by some node being *out*. A "no-good" node Ng is created to record the inconsistency of S, and is used to reject one (called the "culprit") of the "no-good" assumptions A_i by justifying one of the A_i's supporting "out-nodes". So whenever all the no-good assumptions would have previously been in, one of the A_i's out-nodes is in as well, invalidating A_i's supporting justification thus preventing the contradiction from becoming believed.

The only other procedure used by the JTMS is one that approximates CP-justifications as SL-justifications as it is difficult in most circumstances to test the validity of CP-justifications. Doyle [1983] does give a formal specification of how the JTMS *should* consider CP-justifications even if the algorithms described do not achieve this. The mechanics of approximating CP-justifications will not be discussed here and is only considered in passing in §3.3.2. Furthermore, some of the problems of using CP-justifications to record entailment relationships can be seen in §3.3 and §4.3.2.3 which demonstrate that in general it is very difficult to establish entailment relationships purely through examining the dependencies linking propositions.

[3] A directed graph is strongly connected if every pair of nodes is connected i.e. given nodes n and m there is a path from n to m and vice versa.

The algorithms presented by Doyle and Goodwin have a number of design and implementation problems. The most important implementation problem is the inability to deal with unsatisfiable dependency networks which cause incorrect termination before finding solutions, or no termination at all. DDB is also prone to failure in that alternative derivations of a contradiction may exist but are not removed.

The JTMS was initially presented as an algorithm for doing Truth Maintenance and Dependency Directed Backtracking. It was only after this that a formal specification was given, first in terms of generating *grounded admissible expansions* of a network [Doyle 1983] and latterly as modal logics [Brown 1985, Brown and Shoham 1989], operations over lattices [Brown et al 1987, Ginsberg 1988], and most recently using stable sets [Elkan 1990]. No proof of the correctness of Doyle's algorithm has been given and although Goodwin claims such a proof for his algorithm there is in fact an error[4].

Finally, the complexity of such systems has only been briefly investigated. Goodwin (incorrectly) claims a complexity of polynomial degree 2 for updates to a network, while Elkan [1989] has shown, by transformation of propositional satisfiability, that the problem of deciding if a given interpretation is admissible for a given dependency network is NP-complete. In chapters 3 and 4 I will address these issues in depth and present solutions to some of them.

2.1.2 Assumption-Based Truth Maintenance

The design problems raised by de Kleer's attempts to use the JTMS in Qualitative Physics led to the development of *Assumption-Based Truth Maintenance Systems* (ATMS) [de Kleer 1986]. The main problem de Kleer encountered is that the JTMS maintains a single consistent problem solving environment at all times, and that changing contexts through Dependency Directed Backtracking is a time consuming, hit and miss process due to the inability to explicitly select the new target environment. This goes exactly against the type of methodology used by de Kleer in which many choices need to be made and many possible

[4] See Appendix 1 for an example of incorrect behaviour in Goodwin's algorithm.

solutions need to be constructed and examined in parallel [de Kleer 1984].

The ATMS] is based on the familiar structure of nodes and justifications, but instead of using the justifications to assign values of *in* and *out*, each node is assigned a set of *environments*, $\{E_1, ..., E_m\}$[5]. Each environment $E_i = \{A_1, ..., A_p\}$ represents a minimal set of assumptions from which the node in question could be derived. So given the problem solver is currently working in a particular environment E_c, then for every node n with a label $\{E_1, ..., E_m\}$, n is derivable in E_c iff $E_i \subseteq E_c$ for some i.

The ATMS can been seen as a way of simultaneously maintaining a set of JTMS environments but with one important change: the justifications appearing in the ATMS are restricted to having only in-nodes. Assumptions and premises are specifically designated by assigning nodes particular values while contradictions are not explicitly marked as such but are generated by having nodes justify or support the distinguished node $n_\perp$ representing falsity or a contradiction. A premise P is given a value $\{\{\}\}$ indicating P can be derived, i.e. is true, in any problem solving environment. An assumption A is given a value $\{\{A\}\}$ where *A* is a unique assumption identifier indicating that A has been assumed. Obviously if *A* is in the problem solver's current context (i.e. the set of assumptions currently being used) then A is true.

The environments assigned to $n_\perp$ represent contradictory or no-good sets of assumptions: these are used in assigning labels to contingent nodes to prevent contradictory facts being derived. Given a justification

$$(a_1, ..., a_m)_{in} \longrightarrow_{sl} n$$

where each a_i has a label $L_i = \{E_{i,1}, ..., E_{i,m}\}$ the justification is valid in any environment E that supports each a_i, i.e. when

$$\forall i \in [1,m] \exists j . E_{i,j} \subseteq E.$$

In particular, n can be derived in any environment $E \in L$ where

[5] This idea appears at an earlier date but in a less sophisticated form as *contexts* in the *Multiple Belief Reasoner* (MBR) [Martins and Shapiro 1983]. and later in [Martins and Shapiro 1988]

$$L = \{ E_1 \cup ... \cup E_m \mid E_i \in L_i \}.$$

L may contain redundant or superseded environments so L is reduced to minimal environments by removing E_i if $\exists E_j . E_j \subset E_i$. Similarly any no-good environment (i.e. an environment that has a subset that supports $n_\perp$), is removed from L. This results in a label L such that $\forall E \in L . (\nexists NG \in L_{n_\perp} . NG \subseteq E \wedge \nexists E' \in L . E' \subseteq E)$. If a node has more than one justification then the labels they support should be unioned and rechecked for redundancies[6].

In the same way that the JTMS uses Truth Maintenance procedure to update *in/out* values, the ATMS updates labels as new justifications are added. Adding new assumptions or justifications generally leads to the addition of new environments or the expansion of old, while changing derived nodes to premises or marking contradictions will lead to the removal of environments.

Having constructed an interpretation consisting of labels for each node, it may be necessary to reason about or in a particular problem solving context. To actually construct such a context requires a second phase of interpretation. Given the set of assumptions that characterises a particular problem solving environment E_{ps}, each node n is included in the context iff one of its environments is a subset of the problem solving environment, i.e.

$$\exists E_{i \in} L_n . E_i \subseteq E_{ps}.$$

Obviously the whole context may not need to be constructed in order to find a solution i.e. where only the truth or falsity of some nodes relative to a particular context are required.

---

[6] Computationally, if a factorial number of subsumption checks are needed to guarantee minimal labels then it is more efficient to check for redundancies at both a justification and a node level unless the number of redundant environments removed within all dependencies is approximately less than 2N/Av×d where N = number of nodes, Av = average size of label (i.e. number of environments per node) and d = average number of dependencies per node. This assumes a even spread of redundancies throughout the network.

If the environments keep their original order i.e. $L = \{E_{1,1} .. E_{1,n} E_{2,1} .. E_{2,m} ... E_{d,p}\}$ and redundancy is checked from left to right then if $E_{2,m}$ is redundant at a justification level then only n tests (against $E_{1,i}$) are needed to eliminate it at the node level. However, this does not prevent it being involved in up to p! other (unnecessary) checks prior to this.

2.1.2.1 Extensions to the ATMS

De Kleer [1986b] shows how to extend the ATMS so as to be able to construct more expressive relationships. In particular, all the justifications in the basic ATMS are monotonic i.e. contain no antecedents of the form "if α isn't true then ..." but in the extended ATMS it is possible to code such justifications. Most ATMS labellings are capable of generating several specific contexts and the functionality added in the extended ATMS (e-ATMS) helps to restrict the set of possible contexts or select particular ones.

Constraints of the form *choose*$\{A_1, ..., A_n\}$ force all environments to contain at least one of the A_i's. Constraints of the form *ignore*$\{B\}$ means that any environment in a node's label that contains B will be ignored, i.e. not visible to the problem solver, unless B is forced to be included in an environment in order to satisfy some *choose*. E.g. given *choose*$\{A, B\}$, *ignore*$\{B\}$ the preferred environments are those containing A. If A is excluded (because $\{A\}$ is a no-good) then all environments must contain B.

Negation of an assumption can be achieved by forcing a choice between A and $\neg A$ (*choose*$\{A, \neg A\}$) given the extra conditions that an assumption and its negations are contradictory ($\{A,\neg A\}_{in} \longrightarrow_{SL} \bot$) and that assumption A is preferred over the negation (*ignore*$\{\neg A\}$). If and only if A is contradictory does $\neg A$ enter a particular environment. This schema can also be used to encode the negation of any node n by introducing a new node $\neg n$ and ignored assumptions N and $\neg N$ and forcing one to be in every environment. If one node produces a contradiction then the other is included but if neither or both are contradictory then either can be included.

Non-monotonic justifications with in-nodes A and out-nodes B, $A_{in}, B_{out} \longrightarrow_{SL} c$, are transformed into $A \cup \{N\}_{in} \longrightarrow_{SL} c$. Extra assumptions are added to represent the status of the out-nodes: N representing the assumption that none of the out-nodes are actually in the current context, and O is the assumption that some of the out-nodes are in fact in the current context. The constraints *choose*$\{N, O\}$ and *ignore*$\{O\}$ mean that unless there is evidence contradicting N then it will be added to any environment, in preference to O. An extra node o is needed to capture the environments that contain some out-node giving o the intuitive meaning "all the out-nodes are not *out*". For each $b \in B$, the justification $b_{in} \longrightarrow_{SL} o$ is added

to the network as is $O_{in} \longrightarrow_{SL} o$. Finally the justification $\{o, N\}_{in} \longrightarrow_{SL} \bot$ ensures that if any out-node is *in* then N is a contradictory assumption[7]. So given A is contained in the current environment and no $b \in B$ is in the current context then N will be in the context as will c.

This approach is also taken by Dressler [1987] in his "Extended Basis ATMS". Like de Kleer, Dressler adds an assumption (an *out-assumption*, distinguished from normal assumptions) that all the out-nodes in a non-monotonic justification are indeed *out*. Out-assumptions are treated as normal assumptions for the purpose of constructing labels but are also used to create μ-*extensions* of particular environments. A μ-extension of environment E with respect to a set of out-nodes O is a minimal set of assumptions that contain E and includes Out(b) if M is incapable of supporting b or ¬b. The main difference comes in how extensions to environments are actually constructed. Whereas de Kleer uses explicit *choose* and *ignore* constraints to prefer the assumption that the out-nodes are indeed *out*, Dressler gives an explicit special purpose algorithm for generating μ-extensions.

Another interesting extension of the ATMS is the incorporation of mechanisms for dealing with uncertainty. In particular several independent presentations [Laskey and Lehner 1988; Falkenhainer 1987; D'Ambrosio 1987] exist for the idea of using Dempster-Shafer belief theory to assign a numeric value of belief to each node. In particular, given weights assigned to each assumption, the rules for combining evidence allow a single value to be derived from its label.

The last two extensions of the ATMS to cover non-monotonic or uncertain reasoning result in a three (rather than two) phase model of use. Rather than propagating values and constructing a single interpretation from a characterising set of assumptions (as happens with the ordinary ATMS) it is necessary with an extended or non-monotonic ATMS to calculate labels for all nodes, **construct** a characterising set of assumptions given the constraints imposed (either through creating a μ-extension or using *choose* and *ignore*), and finally to

[7] This is a somewhat simplified presentation than the one given by de Kleer. There, an extra node i is included with a justification $A_{in} \longrightarrow_{SL} i$ in order to "gate" the dependency $\{o, N\}_{in} \longrightarrow_{SL} n_{\bot}$ by including i as an in-node, viz $\{o, i, N\}_{in} \longrightarrow_{SL} n_{\bot}$. Also the justifications for generating the contradictions between O and N are overcomplicated and incorrect.

use this to create the interpretation. Similarly, Dempster-Shafer-based ATMSs create a labelling of nodes with assumptions, assign weights to assumptions and then calculate weights for each node based on its label and the weights for assumptions.

Semantics for the ATMS have been provided by theoretical frameworks designed to unify both J- and ATMSs. Explicit algorithms for calculating and updating labels have not been given in the literature but this is less important given the clear semantics and ease of implementation. Obviously no correctness or termination proofs can therefore be given.

Provan [1987] investigates the complexity of using an ATMS in a vision recognition system. He concludes that in trying to find the maximal consistent environments unless there are a large number of small no-goods, a large proportion of the possible environments (numbering 2^n for n assumptions) will be constructed and considered. So for problems with many (partial or complete) solutions the behaviour is exponential. This prompts the development of strategies for considering a single solution at a time. This applies to the process of actually constructing environments and contexts and not to the actual construction of node labels.

Dependency Directed Backtracking can be incorporated within the ATMS [de Kleer 1986d] through the use of disjunctions and *consumers* [de Kleer 1986c], local demons for controlling the addition of justifications to an existing dependency network. This makes it possible for the ATMS to construct the environments in a depth-first rather than a breadth-first manner[8]. More general approaches to controlling the construction of environments exist [Forbus and de Kleer 1988] where the choice of focus (which environment to work with) is the responsibility of the problem solver, not the ATMS.

It might be expected that the lack of non-monotonic justifications in the basic ATMS would lead to a reduction of complexity. It is certainly true that the introduction of non-monotonic justifications and the construction of contexts from environments [de Kleer] or μ-extensions from environments [Dressler] gives rise to a proof of NP-completeness. The basic process of constructing labels itself is also computationally expensive. Either no-goods

[8] This represents a step "back", toward DDB systems that "backtracking without choices" was supposed to have eliminated or made redundant.

are removed from labels as they are generated resulting in |NG| iterations to remove all no-goods (where NG is the set of no-goods) or the no-goods are accumulated to be removed in one pass having constructed all the minimal labels. This approach will result in (possibly) exponential growth in the number and size of environments in node labels as the depth of the node increases.

2.1.3 Logic-Based Truth Maintenance

A different approach to Truth Maintenance, both in terms of objectives and implementation, is the logic-based (LTMS) approach [McAllester 1978, 1980, 1982] based on satisfying clauses. Chronologically it comes between the J- and ATMS but in terms of purpose and style it represents a divergence from the J/ATMS mainstream.

McAllester [1978, p1] states that the purpose of a TMS is to " ... maintain the logical relations among the beliefs that (problem solving) systems manipulate" and to " ... incrementally modify the belief structure when premises are changed ... ". The LTMS's primary function, as with the JTMS, is to provide a consistent interpretation of propositions according to user defined dependencies. The key difference is that the LTMS assumes consistency unless there is evidence to the contrary and uses it to perform "simple propositional deduction" in the form of unit clause resolution. Thus belief values are not assigned on the basis of explicit support and justification, but on the basis of already assigned belief values in conjunction with this assumed consistency.

This gives rise to the following differences:

(a) the LTMS works within a three valued rather than a four-valued logic that underlies the JTMS. Any node is assigned a value of "true", "false", or "unknown" dispensing with the need to have one node represent a proposition and another its negation for there is always a strict relationship between a node and its negation. If π is unknown then so is $\neg\pi$; if π is true then $\neg\pi$ is false and vice versa.

(b) a proposition is deemed to be a premise, assumption, or contingent belief not by the structure of the justifications or by the value of its label but by the way the belief value is assigned, be it asserted or assumed by the user (using an LTMS function) or deduced

by the LTMS;

(c) being a contradiction is not a property that can be ascribed to any particular proposition (or sets of propositions) but describes the state of the TMS when a relationship between some propositions cannot be satisfied and a consistent assignment of value(s) cannot therefore be made.

The propositions of interest are still represented as nodes but the relationships between them are not expressed in terms of support and justification but as disjunctive clauses of the form

$$((P_1. v_1) \text{ or } (P_2. v_2) \text{ or } ... \text{ or } (P_n. v_n))$$

where the P_i's are the related propositions and $v_i \in$ {true, false}. These clauses act as constraints on the values of the nodes, and can be used to represent any formula in propositional logic:

not P	≡	((P. false))
P or Q	≡	((P. true) (Q. true))
P and Q	≡	((P. true))
		((Q. true))
P implies Q	≡	((P. false) (Q. true))

and, for example

(A and B) implies C	≡	((A. false) (B. false) (C. true))

As with the JTMS, the basic action performed by the LTMS is the addition and removal of dependencies and additionally the assertion or assumption of a belief value for a node. When new constraints or values are added, the LTMS deduces any belief value that is necessary to maintain the consistency of the system, i.e. to make sure all the constraints are satisfied, and marks that node as dependent on the other nodes in the clause. When constraints or values are removed, the LTMS checks these dependencies to verify that each dependent value is still implied by the consistency condition and removes any that are not. If a contradiction occurs DDB is used to find the assumptions and premises that underlie it and an attempt is made to remove it by progressively revoking assumptions, by assigning the

opposite value to that assumed. If this is unsuccessful, the LTMS will signal the user that a contradiction has arisen between the identified premises, and will ask the user to choose one to be revoked. Because the assignment of belief values is dynamic the LTMS is also given the ability to detect contradictions that would occur as the result of the assertion of some particular value (P. v), and can construct a clause to deduce the negation (P. ¬v) thus avoiding a contradiction. If the user still wishes to assert (P. v) then a contradiction will occur as normal and DDB will be called to resolve it.

Input	**Deduced Constraint**	**Deduced Values**	**Status of Value**
(Assert (P or Q or R))	((P. true) (Q. true) (R. true))	(P. unknown)	initial value
		(Q. unknown)	initial
		(R. unknown)	initial
(Assert (P. false))	no constraint deduced -	(P. false)	asserted
(Assume (Q. false))	a value is simply assigned	(Q. false)	assumed
		(R. true)	derived

If (R. false) is asserted at some later state, then this value takes precedence over the deduced value and a contradiction occurs as ((P. true)(Q. true)(R. true)) is now unsatisfied. DDB traces the source of the contradiction back to the assertions of (P. false) and (R. false) and the assumption (Q. false). Given the weaker status of (Q. false), this value is revoked and a new value (Q. true) installed. If all the traced values had equal status then it is necessary to appeal to the user to resolve the contradiction.

The LTMS as presented by McAllester represents a shift away from Doyle's general JTMS to a more structured TMS through the assumed consistency and the logical form of the representation of the dependencies between nodes. This can be carried further [McAllester 1980] so that the user does not have to transform a logical dependency into a disjunctive clause to be input into the system. Instead, the logical dependency can be specified in terms of normal logical connectives which are then automatically interpreted as a series of constraints that give the same power as before. This allows the formulae themselves to be assigned a belief value and gives added expressiveness as relationships between nodes can be altered without retracting clauses but by changing the truth value of the connecting formula.

For example, consider the relation (P or Q) previously [McAllester 1978] represented as:

(P or Q) ≡ ((P. true) (Q. true)),

but subsequently [McAllester 1980] as

(P or Q)	≡	(("P or Q". false) (P. true) (Q. true))	(C1)
		(("P or Q". true) (P. false))	(C2)
		(("P or Q". true) (Q. false))	(C3)

In the earlier of the two examples, asserting "P or Q" is true will result in a relationship between "P" and "Q" so that in any interpretation one of "P" or "Q" has to be true. In the second example asserting "P or Q" is true generates a set of clauses links "P", "Q" and "P or Q". This process is necessary to give "P or Q" the intended semantics so that every time "P" or "Q" is true then "P or Q" is true and so on. It is the addition of ("P or Q" .true) that actually asserts that the relationship between P and Q holds, causing C2 and C3 to be satisfied and reducing C1 (functionally) to ((P. true) (Q. true)). Retracting ("P or Q" .true) removes this relationship between P and Q and it is no longer possible to derive a value for P based solely on the value of Q.

This imposition of logical structure culminates in the formal reasoning system RUP ("Reasoning Utility Package") [McAllester 1982] that provides a variety of Truth Maintenance functions within the context of automated reasoning.

2.1.4 Development of TMSs (again)

Chronologically, the initial development of TMS ideas from the JTMS through the LTMS and the *Multiple Belief Reasoner* (MBR) [Martins and Shapiro 1983, 1988] to the ATMS represents a coming full circle as the basic ATMS returns to a more simplistic structure of nodes and justifications compared to those of McAllester and Martins and Shapiro. In another sense the ATMS is the culmination of a move away from the expressiveness and flexibility of the JTMS, through the assumption of consistency (LTMS), the application of a logical framework (LTMS and MBR), to a simple search mechanism. The underlying philosophy seems to have changed from belief through justification, to

validity from premises and finally to values from choices.

2.2 Unifications of TMSs

Many different approaches have been taken in providing semantics for TMSs. Of these approaches, most have concentrated on either the JTMS or the ATMS, no work being done on the LTMS and little work being done on a unified approach to all three. Attempts to provide a unified framework for the A- and JTMS have centred around the use of *lattices* [Brown et al 1987, Ginsberg 1988][9]. One could view non-monotonic ATMSs and the provision of Dependency Directed Backtracking for the ATMS as attempts to unify A- and JTMS style Truth Maintenance but these are only unifications in the sense of trying to provide similar high level functionality. There is no clear correspondence at a detailed level.

Brown et al centre their unification on the structure of justifications. A dependency network is viewed as a set of equations with each node corresponding to a variable or *lattice unknown*. Justifications give rise to conjunctive expressions of the form

$$\alpha_1 \wedge ... \wedge \alpha_n$$

where each α is a *lattice form*: that is recursive combinations of a set of distinguished *situations* (effectively constant, singleton elements of the lattice) and lattice unknowns using the lattice operations $\wedge$, $\vee$ and $\neg$. In a given dependency network each node may have none, one or many justifications that must all be taken account of when assigning that node a value. This is captured by the idea of a *lattice equation* $X = Y$ where X is a lattice unknown (node) and the right hand side is, in theory, any lattice form. In practice the lattice form will be a disjunction of lattice forms corresponding to dependencies:

$$s = d_1 \vee ... \vee d_m.$$

A complete dependency network corresponds to a *lattice equational system*: that is a set of

[9] It is assumed that the reader is familiar with the concept of a lattice as a partial order over a set with: distinguished elements *top* ($\top$) and *bottom* ($\bot$); a *join* (greatest lower bound) operator; a *meet* (least upper bound) operator; and a complement (negation) operator $\neg$ where $a \wedge \neg a = \bot$ and $a \vee \neg a = \top$.

Note - this is in fact the definition of a *complemented lattice* rather than an ordinary lattice, but for this thesis any lattice will be assumed to be complemented unless otherwise stated.

equations in which only a finite set of unknowns appear, and in which each unknown only appears on the left hand side of at most one equation.

Apart from the lattice unknowns, forms, equations and equational systems, there is also the underlying lattice $\mathcal{B}$ built on the set of situations mentioned above, using the operations $\wedge$, $\vee$ and $\neg$.

In this (lattice-based) formulation, Truth Maintenance is the process of finding solutions, a mapping $\Gamma:U\longrightarrow\mathcal{B}$, to a lattice equational system and of updating that solution as equations are added or deleted. The difference between doing ATMS or JTMS-style TM arises from using different underlying lattices $\mathcal{B}$. As noted in §2.1.2 the ATMS uses a restricted set of JTMS justifications so it would be possible, within this restriction, to switch from ATMS to JTMS by recalculating the values for the unknowns using the appropriate lattice. By mapping ATMS assumptions to JTMS premises (i.e. making each assumption letter equivalent to *in* or *out*), and JTMS premises to ATMS assumptions (i.e. making each premise justification $\longrightarrow_{SL}$ n assign an assumption letter) the transition from ATMS to JTMS represents the instantiation of a particular context (i.e. the selection of the context represented by the assumptions mapped to *in*) while the transition from JTMS to ATMS represents a fragmentation of a particular context into a multitude of possible contexts (by being able to selectively assert premises by including the relevant assumption within a particular context). For example, see the transformation below where the assumptions G and H map to *out* while I and J map to *in*.

The JTMS lattice $\mathcal{B}_J$ consists of just two elements $\top$ and $\bot$ corresponding to *in* and *out* with $\top > \bot$. The ATMS lattice $\mathcal{B}_A$ is more complex with elements consisting of ATMS labels, i.e. $\mathcal{B}_A = Pw(Pw(\mathcal{A}))$ where $\mathcal{A}$ is the set of assumptions and for $v \in \mathcal{B}$, $v = \{E_1, ..., E_n\}$ where each E_i is an environment or set of assumptions[10]. For two elements $v,w \in \mathcal{B}$, $v \leq w$ if for each environment $E_i \in v$ there is some environment $F_j \in w$ s.t. $F_j \subseteq E_i$. Obviously $\top \equiv \{\{\}\}$ as $\{\} \subseteq E$ for any $E \subseteq \mathcal{A}$ and $\bot \equiv \{\}$. The

[10] This is similar to Ginberg's [1987] formulation of the ATMS *bi-lattice* in which each node n is assigned a value from $Pw(Pw(\mathcal{A})) \times Pw(Pw(\mathcal{A}))$ where the first element is the set of environments supporting n while the second corresponds to those supporting the negation of n.

difference in lattices carries over to the creation of premises and assumptions: in the JTMS premises are created by including the element *in* ($\equiv \top$) in a node's lattice equation, while in the ATMS the element $\{\{\}\}$ (also $\equiv \top$) is added. Unlike JTMS assumptions which are not fixed but are a transient phenomenon associated with particular labellings or solutions, ATMS assumptions correspond to the inclusion of an environment $\{A_i\}$ in a lattice equation.

Example

Given nodes a, b, c and d and justifications $(a, b)_{in} \longrightarrow_{SL} c$ and $(d)_{in} \longrightarrow_{SL} c$ the equation for c is:

$$c = (a \wedge b) \vee (d)$$

If b is a premise, i.e. $b = \top$, and partial solutions Γ_{JTMS} and Γ_{ATMS} have already been constructed for a and d:

$$\Gamma_{JTMS} = \{(a, out)\ (d, in)\}$$
$$\Gamma_{ATMS} = \{(a, \{\{G\ H\}\ \{I\})\ (d, \{\{J\}\})\}$$

then we can calculate a value for c:

$$\Gamma_{JTMS}(c) = (\Gamma_{JTMS}(a) \wedge \Gamma_{JTMS}(b)) \vee \Gamma_{JTMS}(d) = (out \wedge in) \vee in = out \vee in = in$$
$$\Gamma_{ATMS}(c) = (\Gamma_{ATMS}(a) \wedge \Gamma_{ATMS}(b)) \vee \Gamma_{ATMS}(d) = (\{\{G\ H\}\{I\}\} \wedge \{\{\}\}) \vee \{\{J\}\} =$$
$$\{\{G\ H\}\{I\}\} \vee \{\{J\}\} = \{\{G\ H\}\{I\}\{J\}\}$$

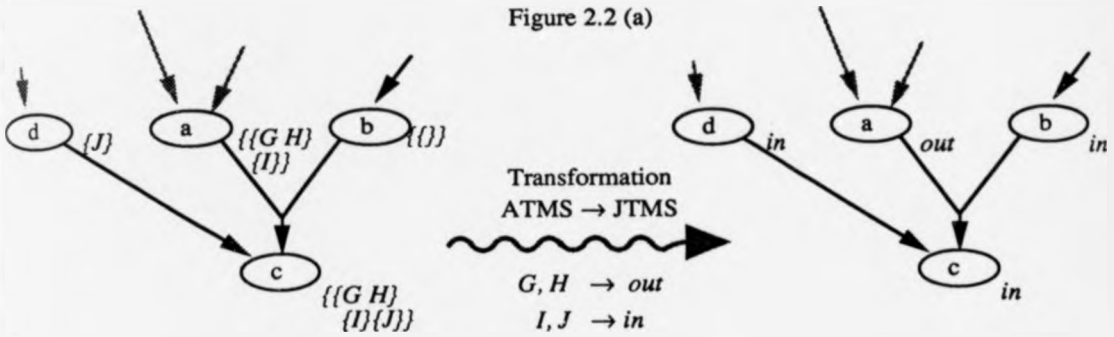


Figure 2.2 (a)

In comparing the results of the lattice equations with that returned by the original algorithms one would expect the same results. This is the case with the JTMS but despite the example above the coding for the ATMS is incorrect, for the ATMS is supposed to produce minimal consistent labels. Simple lattice operations on $\mathcal{B}_A$ are insufficient to produce this. Ginsberg [1988] accounts for no-goods by mapping elements of $\mathcal{B}_A$ to the lattice $\mathcal{B}_A^{NG}$, where NG is a set of no-goods and $\mathcal{B}_A^{NG} = \{y \in \mathcal{B}_A \mid \forall ng \in NG.y \geq ng\}$. I.e. $\mathcal{B}_A^{NG}$ is $\mathcal{B}_A$ with all the inconsistent environments removed: no-goods or supersets of no-goods (which are by definition less than the no-good) are mapped to $\perp$. Redundant environments remain problematic: given labels $v = \{E_1, ..., E_n\}$ and $w = \{E_1, ..., E_n, E'\}$ where $E' \supseteq E_i$ for some i, $v \leq w$ and $w \leq v$. It is necessary to factor $\mathcal{B}_A$ by equivalence classes so that L is replaced by [L], the set of labels generated from L by introducing redundant supersets of the environments in L.

This approach to unification is of interest for it shows how the switch from J to ATMS can be accomplished by simply changing the set of values used to solve the lattice equation

system. However, by insisting on a lattice structure on the values and by restricting the operations to $\neg$, $\vee$, and $\wedge$ (and by assuming an underlying classical logic in the case of Ginsberg) this approach is too restrictive when we want to consider other knowledge representation schemes in Chapter 5.

2.3 Comparison of (Problem Solving) Methodologies

Not only does the lattice-based approach to unification place too many restrictions on the values to be assigned to dependency networks but it fails to bridge the gap between the declarative semantics of the various Truth Maintenance operations and the procedural semantics. From a declarative point of view one would not expect part of the semantics to change as problem solving progresses. However, this is the case when $\mathcal{B}_A$ is mapped to $\mathcal{B}_A^{NG}$ to remove no-good environments from the ATMS lattice. From a procedural point of view the lattice-based approach does not capture how values are asserted and retracted nor the other differences in the approach to problem solving that different TMSs cater for. In this section I shall explore some of these differences.

2.3.1 Premises, Assumptions and Defaults

In addition to having different approaches to the structure of the information passed to the TMS, the systems outlined above have different approaches to the every day meta-level concepts of assumptions, defaults, premises and contradictions. At first glance, it might seem that assumptions, defaults and premises are the same; they all represent the assignment of a truth value to a proposition, be it implicit, e.g. "I assume the sky is blue" or explicit, e.g. "I assume 'the sky is blue' is true". However, there are both qualitative and quantitative distinctions, i.e. differences both in the strength of that assignment and how it is made (and revoked).

Informally a premise is "a statement on which reasoning is based" (dictionary definition), and to assert it is "to declare as true" (dictionary definition) that statement, be it true in the real world, or "a priori" by definition of some particular formal system or theory. Given this definition, it is not possible to change a premise without fundamentally changing

the reasoning system that one is dealing with, and anything derived from it should be held to be true without reservation (unless there is doubt about the soundness of the inference process).

This differs from an assumption which is "taken to be true before there is proof" and can therefore be disproved, or shown to be inconsistent. Anything that is derived from an assumption should be held to be true or false, contingent on the truth or falsity of the assumptions used in its derivation.

A default is also an assignment of truth without proof, but unlike an assumption, a default is only true when some specific condition holds, so assumptions could be viewed as unconditional defaults, or defaults viewed as conditional assumptions. In practice, default conditions are commonly expressed as the failure of some other condition either in terms of explicit negation or by a negation as failure (see §1.1).

Another way of looking at the difference between defaults and assumptions is by using the distinction between cause and evidence [Quine and Ullian, 1970 p6]: assumptions provide a cause for belief but no evidence, whilst defaults provide both cause (by there being a default rule) and evidence (because some condition does not hold). To say that an assumption A should be made on the absence of contradictory information $\neg$A (the approach taken in consistency based non-monotonic logics) is putting the cart before the horse - this is a default rule not an assumption.

Another difference between defaults and assumptions is the way they behave in the presence of contradictions. When a contradiction occurs as a result of an assumption the assumed data should be retracted as there is no evidence for its support, but where a default leads to a contradiction what should be done?

The status or value of an assumption is independent of the values of other propositions and therefore not subject to change due to changes in values of other propositions. The only time it does change is when an assumption is refuted or revoked as the result of some procedure called to restore consistency when a contradiction has arisen. Default conclusions **do** depend on other propositions. When a default is involved in generating a contradiction the default conclusion cannot simply be revoked: there is evidence for a default conclusion.

In order to revoke the default the evidence supporting it must be removed. Additionally, it is possible for a default conclusion to become invalidated without actually removing the default rule that supported it.

With regard to TMSs how are these differences handled? The LTMS does not capture qualitative differences between all premises, assumptions and defaults but simply uses different tags to order propositions for retraction on the basis of strength of the labels. The JTMS does not really capture assumptions as such. What Doyle calls assumptions are merely nodes where the addition of a valid justification will (potentially) remove a contradiction. If assumptions are to be true but defeasible then to assume some proposition π it is necessary to have a node n_π justified with $(n_\Pi)_{out} \longrightarrow_{SL} n_\pi$ where n_Π has an intuitive meaning "it is inconsistent to assume π" and has no attached justifications. Again, this really represents a default rather than an assumption, "if there is no evidence that π is inconsistent then assume π"[11]. This is what the JTMS is good at capturing - default rules for non-monotonic reasoning.

Conversely the ATMS was designed for representing and manipulating assumptions and is therefore good at it, but is bad at representing defaults as shown by the extra apparatus needed to encode non-monotonic dependencies (§2.1.2).

2.3.2 Contradictions and Retractions

One of the main tasks of TMSs is to handle contradictions or inconsistencies. Classically, logical contradictions are *well formed formulae* (wff) of the form $\pi \wedge \neg\pi$ although a contradiction may refer to the presence of both π and $\neg\pi$ in a set of theorems derivable from some set of premises without actually deriving $\pi \wedge \neg\pi$. Any set of premises from which a contradiction can be derived is said to be inconsistent. The use of contradictions in logic is limited to the proof of propositions whose negation would otherwise lead to a contradiction through the use of *reductio ad absurdum* (RAA):

[11] This approach is full of danger as shown in §1.1.1.1 when looking at NML-1 [McDermott and Doyle 1980]. In order to force the JTMS into giving n_Π this intuitive reading additional justifications must be added linking $n_{\neg\pi}$ and n_Π and a contradiction must be set up between n_Π and n_π.

if $\Gamma, \alpha \vdash \beta$ and $\Gamma, \alpha \vdash \neg\beta$ then $\Gamma \vdash \neg\alpha$.

The reasoning for this rule is based on the intuition that propositions can take only one truth value in both formal interpretations and in the world itself. In itself RAA says nothing about belief revision, i.e. how assertions and assumptions should be managed for if α is already asserted to be true (along with Γ) then RAA only leads to more contradiction. TMSs provide facilities for doing just this kind of management of beliefs.

In terms of problem solving and search, it is normally the case that a system is trying to satisfy some goal state through the application of operators that transform the initial state, subject to some set of constraints. This gives rise to the notion of contradiction in a broader sense. A contradiction is a proposition or set of propositions Γ that is judged should not occur or be worthy of further exploration. This can be done without the use of formal logical machinery by simply excluding such states from further computation[12]. Such states may (although not necessarily) contain a logical contradiction but it may be difficult and counter-productive to derive it as such. Thus such non-logical contradictions as "free market capitalism contradicts socialist notions of equal pay for all" is not of the form of a logical contradiction but provides a guide for revising beliefs: if one believes both in free market capitalism and equal pay for all then one may consider revising ones beliefs.

This distinction between logical contradiction as multiple truth assignment to a single proposition, and non-logical contradiction as a guide for eliminating states or revising beliefs, is apparent in the way different TMSs define and deal with contradictions. In the LTMS a network is represented by a set of clauses asserted to be true (potentially to a lesser or greater degree), and a contradiction is defined to be an assignment of values to nodes that does not satisfy every clause. In fact the LTMS will generate a logical contradiction because of the way it performs limited inference (resolution) to derive values. Given a clause (((P or Q). false) (P. true) (Q. true)) and a labelling (V) that assigns values to nodes such that V(P or Q) = true, V(P) = false, the LTMS will have already made V(Q) = true so that the

[12] If however one subscribes to the law of excluded middle ($\pi \vee \neg\pi$) and double negation ($\pi \leftrightarrow \neg\neg\pi$) (not something that Intuitionist or Constructivist logicians would hold with) then implicit in saying Γ is contradictory is the assertion that $\neg\Gamma$ is the case.

assertion ((Q. false)) will assign Q two values.

In the case of the J- and ATMS contradictions are user-defined. In the JTMS individual nodes are marked as contradictions and a labelling becomes inconsistent when a contradiction is labelled *in*. In the ATMS contradictions are represented by a single node $n_\perp$ and contradictory propositions or sets of propositions Γ are indicated by having them support $n_\perp, \Gamma \longrightarrow_{SL} n\perp$[13].

Despite the syntactic differences in the representation of contradictions, all styles of TMS approach retraction in the same way. Backtracking takes place to identify some class of justification that supports the contradiction. Some automatic procedure is used to remove the contradiction or where necessary the user is informed of the contradiction and asked to remove some justification when there is no clear candidate. In the case of the LTMS the backtracking proceeds to nodes whose values have been externally justified and the lowest ranked of these is removed. If two assertions are tied then the user is asked to make a choice [McAllester 1980 p11]. Similarly the JTMS recursively traces the supporting justifications[14] of the contradiction to the underlying assumptions and premises. A random choice of which node to retract is made from the set of assumptions[15]. If no assumptions exist then the user is informed and expected to provide a resolution. In both cases TM is called to propagate any changes arising from the retraction.

The process of backtracking and refutation is less obvious in the ATMS. In fact no reference is made to either in de Kleer's papers[16]. The justifications supporting $n_\perp$ generate the set of no-goods (contradictory environments) and because the environments consist exactly of the assumptions that characterise it there is no need to actually backtrack to

[13] In actual fact the JTMS can use the same scheme as the ATMS with nodes previously marked as contradictions used to support $n_\perp$. Dependency directed backtracking starts at a lower point but is otherwise unchanged.

[14] In the JTMS any node that is *in* has an associated *supporting justification* that is valid (i.e. all the in-nodes are *in* and the out-nodes *out* - these are the *supporting nodes*). An assumption is a node whose supporting justification has a non-empty outlist. A premise is one that has no supporting nodes itself.

[15] It is easy to propose ad hoc criteria for which assumption to revoke in terms of number of other attached dependencies or nodes in the dependency but these have no sound theoretical backing. More dependencies may indicate stability or generality, more nodes may represent specialisation and the number of justifications a node supports may indicate its criticality or entrenchment.

[16] In fact de Kleer originally sets out to remove backtracking altogether.

discover them. Nor does any explicit retraction actually take place: no justifications are added to or removed from the dependency network. However it is necessary to remove all no-goods and their supersets from all node labels (retraction of environments). Rather than checking each label it is only necessary to check for where the environment is known to occur ("forward-tracking"). If {A B C} is known to be inconsistent then only the nodes in c = *desc*(A)∩*desc*(B)∩*desc*(C) need be checked[17]. Furthermore if {A B C} is a subset of an existing no-good {A B C D} then c can be refined even further and only c = [*desc*(A)∩*desc*(B)∩*desc*(C)]/*desc*(D) need be checked. If a non-monotonic ATMS is being used it would be necessary to invoke TM to recalculate labels rather than simply performing subset tests. In this case it is even more relevant to use forward-tracking as the cost of recalculation is higher than for simple testing.

From this it can be seen that the process of retracting contradictions is inherently cyclic through the process of backtracking and recalculation. It is this cycle that can lead to much work being done as several passes are needed to remove a set of contradictions. It is this same circularity that leads to the computational problems of default logics as we shall see in Chapter 5.

The important point to make in this section is that the user needs to have the ability to specifically exclude particular interpretations through the avoidance of user defined "contradictions". Additionally, the retraction process often needs to call on the user to make choices about the refutation of assumptions. These refutations can be seen as meta-level operations in that they change the underlying network (and the theory that it instantiates) that is being interpreted. This is not actually the case in the ATMS as only the labelling is changed. However, as we shall see in the next section, this process takes place through a secondary interpretation of the ATMS labelling when particular contexts are constructed from a particular set of assumptions or environment. Reichgelt [1988] is supporting this claim when he proposes an architecture based on an inference engine, a knowledge base and a knowledge base manager. The knowledge base is equipped with TM facilities and it is the

[17] The function *desc* returns the *descendants* of a node as defined in §3.2.1.

role of the knowledge base manager to propose assumptions and instantiations of defaults. In the case of TMSs the user (be it a human or machine reasoner) needs to have access to the designation and removal of contradictions

2.3.3 Traditional Interpretation Construction and Search Strategies

In reviewing existing TMSs, three levels of operation can be discerned: basic truth maintenance, automatically invoked (network restructuring) procedures, and user-initiated or involved procedures. The most basic of these tasks is the result of the coupling of a problem solver with a TMS - the problem solver generates propositions and inferences that are subsequently stored in the TMS as a dependency network. As nodes and justifications are added or removed the labelling of the network is updated. Automatic network restructuring is invoked when particular standard events occur in the network due to the addition or removal of nodes and/or justifications and subsequent changes in the labelling. Finally user initiated or involved procedures are invoked when a certain, network specific state has been reached and the problem solver needs to access information contained in the structure of the network or in the values of the labelling.

The three levels of functionality can be broadly categorised into three levels of interpretation or semantic activity. The first level of functionality (Truth Maintenance) provides basic justification-based belief revision. Changes to the network, i.e. asserted facts or relationships, result in changes of belief in the network and these changes are propagated as if inferences were being made. The second level (automatic network update) corresponds to maintaining consistency[18] of the network and its interpretation. When a contradiction (a general, well specified event) occurs, steps are taken to restore consistency in a standard, pre-defined way that holds for all instances of a particular network. Such procedures represent meta level inference rules or strategies such as reductio ad absurdum - if α and β together generate a contradiction then assert the negation (i.e. remove) one of the assertions that led to the contradiction. The third level (user interactions) represents meta-level

[18] or potentially other high level characteristics

reasoning on the part of the user, acting either to favour or construct particular interpretations of the network. This can be seen in the positive choice of particular interpretations which occurs through secondary interpretation of the ATMS, corresponding to picking the interpretation associated with a base set of assumptions and in the action of *choose* or *ignore* constraints. It also occurs in the disambiguation of networks if the user is involved in explicitly choosing how contradictions are to be resolved in the J- and LTMSs. Meta-level reasoning also occurs in the restriction (negative choice) of possible interpretations through the designation of contradictions, either through adding no-goods to the ATMS, or the marking of particular JTMS nodes.

These three levels of interpretation correspond to different levels of competence and/or complexity in existing TMS systems. The LTMS operates within a single interpretation and operates very much on the first level, propagating belief values and filling in values that would otherwise lead to contradictions. This indicates a predisposition towards consistent interpretations. Automatic procedures exist to track the sources of contradictions but there is no notion of a default inference and therefore no procedure to resolve them, beyond the simple notion of a user supplied ranking of assertions. User involvement in selecting interpretations is limited to resolving contradictions through choosing which premises to retain and interpretations can only be excluded through the addition of propositions and justifications to the network.

The JTMS is also a single interpretation system but operates in more complex ways on more levels than the LTMS. Non-monotonic inferences can be modelled and Dependency Directed Backtracking (DDB) is capable of deciding how consistency should be restored when non-monotonic inferences result in contradictions. By having control over what is designated a contradiction, the user has control over the invocation of DDB.

Finally the ATMS is the most complex of the three types of system examined. It maintains multiple interpretations simultaneously, and spawns new interpretations when new assumptions and justifications are added. User intervention is required to select a particular interpretation, either by the addition of contradictions leading to the exclusion of interpretations and/or the choice of a characterising set of assumptions. Consistency is

maintained partly through the assertion of *no-goods* and partly through the propagation of belief values, as this is done with reference to the set of no-goods.

It would be wrong to say that each system operates exclusively at set levels with the LTMS at level 1; the JTMS at levels 1 and 2; and the ATMS at levels 1,2 and 3. However, the facilities at each level correspond directly to the type of search strategy for which a particular system is best suited as shown in the next section. Not only do these three levels provide an insight into the competence of TMSs in various tasks that form the basic criteria for judging general belief revision systems, but these levels provide a starting point for the definition of the general framework of Interpreted Dependency Networks.

In designing a particular class of belief revision systems, it will be necessary to define the elements of a type of dependency network, the values that could be assigned to those elements, and the basic TM procedures for calculating those values. On top of this a designer may wish to add procedures for performing generic high level problem solving tasks, such as contradiction resolution and checking entailment relations. It is up to the user of such a system to provide the data that generates a particular network and to manage the assertion and retraction of that information, e.g. through the selection of assumptions and the switching of contexts.

2.3.4 Gardenfors' Postulates and TMSs

If TMSs are to be considered belief revision mechanisms, how well do they comply with Gardenfors' postulates? In order to answer this, the notion of a belief state must be adequately defined. This must wait until Chapter 3 where TMSs are reconstructed as Interpreted Dependency Networks. To preview the results, as one would expect of systems based on a foundations approach, TMSs do not satisfy some (or many, depending on the type of TMS) of the postulates. It is interesting to note that it is the non-monotonic dependencies of the JTMS which prevents it from satisfying many of the postulates and it is the lack of such dependencies in the ATMS which means it satisfies more.

2.4 Applications of TMSs

The difference in the basic network, interpretation and procedures of a particular TMS are a reflection of the underlying ideas of consistency, contradiction and assumption. These in turn are a reflection of the underlying problem solving methodology for which a TMS was developed. This in turn is usually reflected in the kinds of applications for which a particular TMS is used.

As stated at the beginning of this chapter, the original TMSs were developed to aid the exploration of a search space. An obvious distinction arises from the difference between recording a single problem solving state and multiple states. Given the state-related notion of assumption (a JTMS assumption is defined by a particular type of dependency in a particular type of interpretation) and the difficulty that this provides in changing states, the first is more suited to a depth-first search to find a single (or possibly a few) solution, backtracking when an inconsistency is found.

The indexical, stable nature of assumptions in multiple context TMSs is more suited to a breadth-first approach. All possible solutions are developed in parallel and it is only in the secondary interpretation phase when a number of possible interpretations have been eliminated through the assertion of no-good sets of assumptions that particular interpretations are constructed. This is done in order to minimise the number of complete or partial interpretations that have to be constructed. However, it should be noted that the development of DDB and focusing for the ATMS has muddied this distinction as it is now possible to explore a limited breadth of the search space to some depth with an extended ATMS.

This categorisation is no chance result of the choice of structure for interpretation and assumption. Rather it was the desire to pursue these problem solving strategies that led to those choices being made. De Kleer [1984 p79] explicitly states this when he says

> "Qualitative reasoning and constraint languages both require making choices among alternatives. (Existing TMSs) have proven to be woefully inadequate for even the simplest qualitative reasoning tasks .. they are intrinsically incapable of working with multiple contradictory choices at once - something one needs to do all the time in

qualitative reasoning."

In de Kleer's approach to constraint satisfaction [1986 p134-137] the direct connection between search and problem solving can be seen. The problem solver has to choose values for variables in constraints and each choice (assumed value for a variable) represents a branching at some vertex in a search tree. DDB provides facilities for moving (vertically) up and down branches. The ATMS provides efficient mechanisms for recording choices thus making it easy to traverse the tree (horizontally) across branches.

The distinction between ATMS and JTMS as instantiations of different search strategies is borne out by some of the tasks within particular domains to which they have been applied.

2.4.1 Application Domains

Vision

A good example of different tasks needing different TMSs comes in the domain of vision and scene analysis.

Herman and Kanade [1986] are interested in building a single model from a series of wire frame descriptions generated from pictures of the same scene but from different angles. Assumptions are made about the structure of occluded regions and these assumptions must be updated as the scene changes. This requires a JTMS-style, single context, dependency network.

Bowen and Mayhem [1987] are also interested in generating scene interpretations from a pair of stereo wire frame descriptions. The processing is less incremental with data being added to the model as is the case with Herman and Kanade. Rather than seek a single consistent interpretation and backtrack in the case of error, the static nature of the data means that it is possible (and more efficient) to

> " ... follow the development of alternative solutions in parallel, and compare final solutions to choose the best".

This would just not be possible for Herman and Kanade. For this reason a multiple context TMS similar in style to the ATMS is used.

Provan [1987a, 1987b] is interested in the recognition of complex objects within a particular scene. Given there are many possible objects or combinations of objects that could form part of the desired complex object, Provan states that it is natural to pursue a multiple context approach. However, given the combinatorial explosion that naturally occurs in such problems he finds that the ATMS is unable to cope. Here we see the natural breadth-first approach of the ATMS needing to be altered to provide a narrow search by focusing the behaviour of the TMS.

Problem Solving Architectures

At a higher level of generality, different TMSs have been used to augment particular problem solving strategies or methodologies. A traditional rule-based approach to problem solving can be viewed as a search space requiring a single context thus more suited to a depth-first approach[19] Given this, it is unsurprising to find justification-based TMSs used to provide control over the problem solving context [de Kleer et al, 1977] and to provide non-monotonic IF-THEN-UNLESS versions of standard IF-THEN rules [Schaefer et al, 1986].

An obvious example of a TMS tied to a methodology or representation is that of the LTMS (which has not seen any published applications, perhaps because of this tie). Here the target knowledge representation is deeply entwined with the TMS and it should come as no surprise that the LTMS forms the basis of a theorem proving system [McAllester, 1982a, 1982b].

Problem solving that requires an explicit notion of state is one that is well suited to an ATMS approach rather than a JTMS one, given the former has some notion of different states within a given interpretation which the latter does not possess. This has led to the development of KEE-Worlds for the representation of actions or state changes [Morris and Nado, 1986]. This is a general mechanism for problem solving but is obviously very useful in the context of planning or scheduling [Filman 1988].

[19] Obviously different search strategies can be employed with regard to which rule is executed (or fired) or which goal is expanded, but by concentrating on a single context the overall strategy is essentially depth-first.

Planning and Others

Planning has proven to be a fruitful application area for the ATMS, given its multiple context facilities. Multiple flight scheduling [Mott et al 1988] (given constraints on the number and types of flights that must be flown and the available time, aircraft and personnel) and personal flight planning [Padala et al 1986] (given constraints on flight availability, cost, time and destination) have been treated in this way.

Diagnosis has also seen the application of the ATMS [de Kleer and Williams 1986]. Again, the multiple contexts allow the comparison of competing diagnosis and the indexical natural of ATMS assumptions (each component is assumed to be working thus generating contradictions and pinpointing the failed components) allows for multiple faults to be diagnosed at a single pass.

Other sundry applications of TMSs include user modelling [Jones 1987], logic programming [Flann et al, 1987; Drakos 1987], circuit design [de Kleer and Sussman, 1980], and Qualitative Reasoning, the application that forced de Kleer to write the ATMS [de Kleer and Brown, 1984].

2.5 Conclusion

The starting point of this chapter was that a foundations approach to belief revision combined with an explicit network-based representation of inferences would produce a competent belief revision system. Truth Maintenance Systems in general take a foundations approach to belief revision based on a explicit representation of inferences and in most systems these inferences are represented as an explicit network. However, the review in this chapter shows that this is not in itself sufficient to produce a general purpose belief revision system. In particular it can be seen how TMSs developed as a response to different problem solving tasks, methodologies, and application domains. The result of this is that each type of network representation is limited in the kinds of inferences in can represent and the kinds of beliefs that can be managed.

Furthermore, although existing unifications of TMSs can account for the similarity at the basic level of the network and the labellings, this is insufficient to capture the operation

of the TMS as a whole. In particular supposedly common notions of assumption, default, contradiction and interpretation turn out to have specific, individually tailored definitions and uses. This in turn has implications for the way each type of TMS constructs and manipulates a network in a particular problem solving example or domain.

Given these problems with existing TMSs and their unifications, it appears that a single general belief revision system would be inadequate. What is needed is a general way of specifying foundation-based belief revision systems. One of the major contributions of this thesis is in providing just such a framework in the form of Interpreted Dependency Networks.

The following chapter does this by proposing a network structure capable of supporting a variety of different interpretations along with definitions of what constitutes a valid interpretation. This can be viewed as analogous to defining a logical language along with its semantics and this and other semantic issues will be explored. With regard to this chapter, as IDNs are a generalisation of existing TMSs, it will be shown how existing systems fit into this framework. With regard to Chapter 1, Gardenfors' postulates will be examined and it will be shown where Gardenfors' postulates fail to apply to this framework, and why.

In broad terms, Chapter 1 has produced a general set of criteria, from examination of existing work, that one might want a belief revision system to fulfill. This chapter has examined a set of existing systems (TMSs) that come close to satisfying those criteria (but are not capable in themselves of acting as general purpose belief revision systems) in order to identify common structure and functionality. In the following chapter this commonality will be used in defining a general approach to belief revision. As a side effect, it will be possible to: overcome some existing problems with TMS implementations; provide general algorithms for constructing TMS interpretations; and embed information from a problem solving system in a TMS-like belief revision system.

CHAPTER 3

Interpreted Dependency Networks

Truth Maintenance Systems provide one way of managing belief revision: they provide a foundations approach to maintaining a proposition's level of belief. It is only through changes to the arguments supporting a proposition that its value can change. However, in order to be able to revise beliefs, one must also be capable of representing those beliefs and the links between them in a uniform and meaningful manner.

This chapter will do three things in order to promote the use of Interpreted Dependency Networks (IDNs) as belief revision and representation systems:

- §3.1 shows why TMSs are flawed as representation mechanisms despite their capabilities as (potential) belief revision systems:
- §3.2 defines Interpreted Dependency Networks as a way of generalising TMSs in such a way as to give it a sound theoretical basis: and
- §3.3 reconstructs existing TMSs to show how their functionality can be captured within the framework of IDNs.

As a side effect, having given a formal semantics to IDNs and their interpretations, the application of Gardenfors' postulates for belief revision can be examined in a more critical way.

3.1 TMSs as Representation Mechanisms

The review in Chapter 2 suggests that TMSs can be viewed as belief revision mechanisms, despite problems in implementation, flexibility etc. These limitations would mean that they are not ideal as representation mechanisms but could provide a foundation upon which to construct a more general approach. However, as this section will show, there are more fundamental flaws in the conception of TMSs that mean they cannot function as belief revision systems.

3.1.1 Existing TMSs

Up to this point existing TMSs and their behaviour have been critically examined. Despite this, it is worth reiterating their motivation and uses before going on to consider their use as a general modelling mechanism.

The primary use of TMSs is as a component in a system that searches for solutions to problems or queries. Traditionally the TMS does not add anything to the capabilities of a problem solver, save in efficiency, and serves no purpose as a "stand alone" application. Each TMS performs a specialised function within a problem solving methodology. Doyle's JTMS provides Dependency Directed Backtracking [Stallman and Sussman 1976] enabling defeasible inferences to be made and automatically withdrawn. As such it provides a way of implementing depth-first search through a series of defaults or assumptions. De Kleer's ATMS provides a means for recording the derivability of propositions from sets of assumptions. The initial motivation was to provide an efficient way of reducing a search space by the prevention of exploration of areas that could not lead to a solution.

The high level task that these procedures perform, as opposed to how they operate, is to maintain one or more consistent problem solving states. In the case of the JTMS, consistency is maintained by revoking one of a set of contradictory assumptions; McAllester's LTMS selectively revokes propositions according to a weighting scheme indicating the relative strengths of defaults versus assumptions versus premises; and in the ATMS, derivations are not allowed under contradictory sets of assumptions.

In each case, the TMS acts as an intelligent cache of information derived by the problem solver. New information and a record of their derivations is passed from problem solver to TMS, and information about the status of propositions is passed from TMS to problem solver. The TMS acts as an intelligent record of problem solving activity providing a more structured history than a simple transcript.

As originally put forward, TMSs are not representation mechanisms per se. They were not proposed as long term stores of problem solving information. At the start of a session the TMS contains no information and it is only as the problem solver derives information that it is passed to the TMS. The network of justifications grows incrementally as does the interpretation or assignment of values within the network - justifications are added and the interpretation is revised accordingly, more justifications are added and the interpretation is revised again and so forth.

If we want to use TMSs or the dependency networks on which they are based as tools for recording and manipulating long term information then we must alter our conception of what constitutes a dependency network and how we use it.

3.1.2 TMSs as Modelling Tools

In this thesis I advocate the use of dependency networks as a knowledge representation technique, and as such it is necessary for them to have a fixed declarative semantics. It must be possible to interpret any network independently of a program that constructs interpretations. Furthermore, if the dependency network is used to represent long term declarative knowledge then the process of interpretation will not be incremental, as is the case with TMSs. We must be able to construct interpretations of arbitrary networks from scratch, without regard to the incremental behaviour that is usually seen in TMS/problem solving interaction.

In particular, the interpretations should not be sensitive to the order in which the elemets of the network were added - this implicit temporal information will not be part of the stored network representation. If the temporal order is important[1], it should be explicitly coded into the network and/or interpretation structures. This could be done, for example, by explicitly storing temporal information as part of a proposition attached to the node. Alternatively the temporal information could become part of the interpretation so each node might have many values, each flagged with a particular time stamp. The latter approach is analogous to storing values for quantified expressions as demonstrated in §5.2.4 except that in this case variables represent the same object at different points in time (rather than representing different objects or individuals).

For example, Morris [1987] uses the JTMS as a tool for representing defaults and directly codes implications as dependencies. To actually construct an interpretation of the information using the JTMS one would have to pass each statement in turn into the system. Not only is this time consuming, but more importantly, because of the procedural nature of

[1] It is easy to imagine systems where temporal information is used in determining how information is to be interpreted.

the JTMS (it is only a program) and the fact that it can have multiple interpretations, different orders of instantiation of dependencies gives rise to different interpretations. For example, the following operations:

operations	**network**		**interpretation**
	nodes (N)	**dependencies (D)**	
assert $a_{out} \longrightarrow_{SL} b$	{a, b}	$\{a_{out} \longrightarrow_{SL} b\}$	((a *out*) (b *in*))
assert $b_{out} \longrightarrow_{SL} a$	{a, b}	$\{a_{out} \longrightarrow_{SL} b, b_{out} \longrightarrow_{SL} a\}$	((a *out*) (b *in*))

gives us a final interpretation where b is believed and a is not. But the same operations in a different order:

operations	**network**		**interpretation**
	nodes (N)	**dependencies (D)**	
assert $b_{out} \longrightarrow_{SL} a$	{a, b}	$\{b_{out} \longrightarrow_{SL} a\}$	((a *in*) (b *out*))
assert $a_{out} \longrightarrow_{SL} b$	{a, b}	$\{b_{out} \longrightarrow_{SL} a, a_{out} \longrightarrow_{SL} b\}$	((a *in*) (b *out*))

gives the opposite result, a is believed but b is not. This is particularly important because this type of example routinely occurs when dealing with interacting defaults.

The example shows how problems may arise in incremental updates, but this is not an argument against incremental behaviour per se - this thesis looks at this kind of behaviour in §4.3.1.1. This is really an argument for saying there must be some criteria for establishing what are valid interpretations for a network. In the previous example we need to know that there are two or more solutions.

If IDNs are to be a useful representation and revision mechanism then there must be some principled way of constructing interpretations that is independent of how the network is temporally constructed. If it is necessary to have a history of the sequence of actions that led to the creation of a network in order to interpret that network, then the network has no

meaning, independent of the history. It becomes impossible to give a simple, declarative semantics to the network structures.

3.1.3 Intuitive Semantics of Support

The intuitive (procedural) semantics of dependency networks as used by TMSs is clear. The network represents a *record* of inferences made by the problem solver. The values assigned to nodes reflects the system's current "belief"[2] in the proposition attached to the node. In the JTMS this belief is supported by only one justification, (although there may be many justifications attached to a single node) whereas in the ATMS the belief is calculated from all the possible justifications. The values are updated according to how the validity of the inferences change over time, effectively adding or subtracting support for a proposition.

If networks used by IDNs are not a record of inferences made by a problem solver and the inferences' current validity or status, what are they? Intuitively, the nodes represent propositions (both in the naive sense of a statement and in the logical sense of an irreducible formula) and the dependencies represent or **act** as potential inferences. As proposed in Chapter 1, it is this set of inferences that can be used to capture a particular theory. The semantics of these inferences is determined by the functions or algorithms that determine how the values of the antecedents of a dependency support the consequent. These functions correspond to a theory or method of belief combination. A valuation corresponds to a Herbrand model in that there are no "external" semantics in terms of what is denoted by the propositions attached to nodes or the predicates they may contain. This semantics must be supplied by the user of the system.

The main point behind TMSs and IDNs is that the value assigned to each node or proposition should reflect the dependencies that support it. A node cannot be assigned a value, except a null value corresponding to "no other value is supported or justified", unless there is some reason for that assignment, even if it is only a dependency representing the

[2] What constitutes a "belief" for different systems depends on the particular TMS - e.g. for the JTMS "belief" is just that, a proposition is either believed or not, but for the ATMS "belief" corresponds to those worlds in which a proposition is believed, with any other world corresponding to a lack of belief.

justification or reasoning "I am going to assign this node this particular value". We want to exclude valuations that do not have this characteristic (i.e. those interpretations where a node's value does not reflect the level of justifications supporting it) in the same way that classical interpretations of logic cannot have an interpretation I where $I(a) = 0$, $I(b) = 0$, and $I(a \vee b) = 1$. The semantics of the dependencies through their associated functions maintain an internal consistency.

One can think of dependency networks as a kind of propositional formula with root nodes, i.e. nodes that are the consequent of no dependencies, representing propositional letters, dependencies representing connectives and interior nodes representing compound formulae. The construction of interpretations for a network corresponds to the construction of a truth table with assignment of values to root nodes (analogously the propositional letters) and the calculation of values for dependent nodes (compound formulae) being done according to the rules governing the support of dependencies (connectives).

3.2 IDN Definition

Having established the need for a general dependency network representation with a declarative semantics in the preceding section, this section will provide just that. First the network structure and concepts will be defined and then a formal treatment of the (informal) semantics of supporting dependencies will be given. The combination of the dependency network and the interpretation mechanism together form the basic Interpreted Dependency Network and it is this definition of IDNs that forms the major contribution of the thesis. Without this definition there would be no context for the rest of the work contained within this thesis.

3.2.1 IDN Definition: The Dependency Network

The formal definition of a basic *Interpreted Dependency Network* (IDN) comprises:

- a network **<N,D>** where **N** represents a set of propositions $\mathcal{L}$ and **D** is a set of dependencies of various types defined over **N**; and
- an interpretation structure (covered in §3.2.2) consisting of an arbitrary set of values $\mathcal{V}$ including a null value *nil*, and a set of *summation functions* $\mathcal{S}$ that govern the assignment of values from $\mathcal{V}$ to nodes in **N**.

The attachment of propositions to particular nodes is recorded by the mappings $P:\mathbf{N} \rightarrow L$ and its inverse $node:L \rightarrow \mathbf{N}$.

Each dependency belongs to a given type and we can partition D accordingly, i.e.

$D = D_1 \cup ... \cup D_p$ where $d \in D_i$ is of type i and the D_i's are pairwise disjoint.

Each dependency d is of the form $(A_1, ..., A_q, c) \in Pw(\mathbf{N}) \times ... \times Pw(\mathbf{N}) \times \mathbf{N}$. Each A_j is a set of *antecedent* nodes of a given type and the *antecedents* of d are $ants(d) = A_1 \cup ... \cup A_q$. A node being a particular type of antecedent is a function of the structure of the dependency and is not a property of the node itself, and therefore a single node can be of many different antecedent types depending on the dependencies within which it occurs. The node c is the *consequent* of d, i.e. $cons(d) = c$ and we say that that c is *supported* by d and d *supports* c, or alternatively that d is *attached* to c. Similarly, the antecedents of d are said to *support* d.

The function $D:\mathbf{N} \longrightarrow Pw(\mathbf{D})$ returns all the dependencies that support, or are attached to, a particular node. The *parents* of a node n are the antecedents of all the dependencies supporting n: $pars(n) = \bigcup_{d \in D(n)} ants(d)$. Analogously, $C:\mathbf{N} \longrightarrow Pw(\mathbf{D})$ returns all the dependencies that are supported by a particular node and the *children* of a node n are the consequents of all the dependencies supported by n: $chld(n) = \bigcup_{d \in C(n)} cons(d)$. The transitive closures of D and C are the *ancestors* (*ancs*) and *descendants* (*desc*) of a node respectively.

The number of types of dependencies and the number of types of antecedents for each type of dependency is unrestricted (providing it is finite), and is determined by the IDN designer and influenced by the type of information the IDN is being used to capture. For convenience I define a function $t:\mathbf{D} \times Pw(\mathbf{N}) \longrightarrow \{0, 1, 2, ...\} \times \{0, 1, 2, ...\}$ that returns the

type of a given dependency or set of antecedent nodes within the context of a particular dependency. For example, for a dependency of type i, $t(d, \varnothing) = (i, 0)$ which I shall abbreviate to i. For a set A of antecedent nodes, if A is of type j within dependency d (which is in turn of type i) then $t(d, A) = (i, j)$. Where the context of A is apparent, I will write $t(A)$ instead of $t(d, A)$ and refer to the type of A as j rather than (i, j).

Rather than write dependencies as n-tuples, if d is of type i, I shall write

$d = A_1, \ldots, A_q \longrightarrow_i c$ for $d = (A_1, \ldots, A_q, c)$

If A_j is empty for some j then I will omit it from the dependency, similarly if A_j contains just a single element a, then I shall write a_j instead of A_j or $\{a\}_j$. When actually defining the type of dependencies and antecedents for given IDN systems the numbered types may be replaced by other (unique) identifiers.

For example, rather than defining JTMS SL-dependencies to be type 1 dependencies whose type 1 antecedents correspond to in-nodes and whose type 2 nodes are out-nodes, they could be "SL" type dependencies with "in" type in-nodes and "out" type out-nodes. e.g.

$d1 = (\{a\ b\ c\}\ \{d\}\ f) = \{a\ b\ c\}_1\{d\}_2 \longrightarrow_1 f = \{a\ b\ c\}_{in} d_{out} \longrightarrow_{SL} f$

$d2 = (\{\}\ \{a\ c\}\ g) = \{a\ c\}_{out} \longrightarrow_{SL} g$

This then is the basic structure of the network: a set of nodes with dependencies (of different types) supporting them. The dependencies attached to a node are supposed to provide some kind of justification for a given level of belief in the proposition represented by that node. This support is provided through the interpretation structure as shown below.

3.2.2 IDN Definition: The Interpretation

The basic idea behind IDNs is that the belief value assigned to each node is a function of the belief values associated with each dependency. For example, if a single node were supported by two justifications, one of which had an interpretation saying "believe this proposition (i.e. the proposition $P(N)$ attached to the node) to be true" while another, independently had the interpretation "there is no evidence for believing this proposition" then one might expect the node to have a value "believe this proposition" as the first justification provides ecidence for believing it. Alternatively the same case might have a different interpretation: the

justifications could be independent and individually sufficient reasons for believing a proposition and that belief could be statistical in nature. In this case, a value of 0.5 for the first dependency combined with a value of 0.3 for the second would result in a conditional probability for the node (given either dependency were to be true) of $1 - (1 - 0.5) \times (1 - 0.3)$.

Which particular theory of belief representation and combination is used for a particular network is determined by two things: the values assigned to nodes; and the functions that determine how those values are combined at various levels within the network structures:

- at the node level there is a single summation function $S^{\mathbf{N}}$ that determines how the support provided by each dependency, in the form of belief values, are combined;
- at the dependency level, for every type of dependency there is a summation function $S_i^{\mathbf{D}}$ that determines how values from the various sets of different type antecedent are to be combined; and

  at the antecedent level, for every set of typed antecedents j there is a summation function $S_{i,j}^{\mathbf{A}}$ that determines how values of each antecedent are to be combined.

The summation functions form a three levelled hierarchy of functions which could be thought of as a generalisation of an AND/OR graph, or of a disjunctive normal form. At the node level, $S^{\mathbf{N}}$ corresponds to a disjunction over the support provided by the attached dependencies. For each type i of dependency, $S_i^{\mathbf{D}}$ corresponds to a conjunction over the support provided by each class of antecedent $A_{i,j}$, where the antecedent class support is calculated using $S_{i,j}^{\mathbf{A}}$. It should be noted that the disjunctive normal form is only a metaphor for the structure provided by the summation functions, and although dependency networks will frequently have this structure, it is by no means necessary.

The actual semantics of the network elements is provided by the summation functions, and it is these functions that will determine if a network is interpreted as an AND/OR graph, or an OR/AND graph or even a MAY-BE/PERHAPS/MAY-BE-NOT network (given an appropriate set of summation functions, if one existed)! What is common however is that at each level there is a combination and distillation of belief values: values for antecedents are grouped by dependency and antecedent type before being combined to a single value for each antecedent type (for a given dependency), which are then in turn combined into a single

value for each dependency. These dependency values are in turn combined into a single value for each node. What started as a set of values for antecedent nodes have been manipulated, combined and reduced to just a single value.

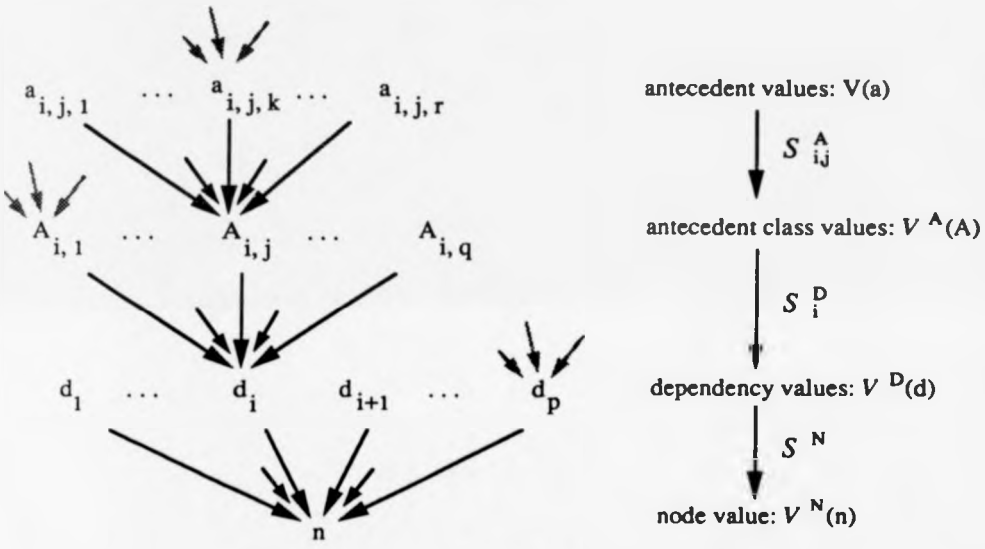


Figure 3.2.2 (a)

Both the distillation of values and the disjunctive normal form can be seen in Doyle's [1979 p240] seminal TMS paper where an SL-justification "is valid (i.e. supports a value *in*) if and only if each node in its *in*list is *in*, **and** (added emphasis) each node in its *out*list is *out*". A node is only *in* iff at least one of its justifications is *in*[3].

[3] There is the added proviso that the *in* justification should not be self-supporting. This is captured by the kind of interpretations allowed for the network as shown later.

This then is the basics of the semantics of the network. The summation functions determine the meaning of the dependencies attached to nodes, and give a set of rules for saying how values might be combined. But a declarative statement is also needed to say when the semantics of the dependency structure has been adhered to and the resulting interpretation is a valid one.

3.2.3 Formal Semantics of Support

Formally, a labelling of the nodes in **N**, or more specifically a *valuation* V *over a network* **<N,D>** is an assignment of values to nodes V: $\mathbf{N} \rightarrow \mathcal{V}$. Given a fixed ordering on **N** we can write V as an n-tuple $(V(node_1), ..., V(node_{|N|}))$ and the set of all possible valuations is $\mathbf{V}^{|N|}$.

We are now in a position to write formal conditions to capture the idea that each node's value, with respect to a particular valuation, must correspond to the value supported by the dependencies attached to it.

The value of a given node n under a valuation V, written $V^{\mathbf{N}}(n, V)$, is defined to be the sum (as defined by the summation functions as outlined in the preceding section) of the values of the dependencies, $V^{\mathbf{D}}(d_i, V)$, for each dependency attached to n:

$V^{\mathbf{N}}$: $\mathbf{N} \times \mathbf{V}^{|N|} \longrightarrow \mathcal{V}$;

$(n,V) \longmapsto S^{\mathbf{N}}(V^{\mathbf{D}}(d_1, V), ..., V^{\mathbf{D}}(d_m, V))$

where $d_i \in D(n)$

The value of each dependency d, $V^{\mathbf{D}}(d, V)$, is similarly defined with the value being equal to the sum of the values, $V^{\mathbf{A}}(A_{i,j}, V)$, of the antecedent classes:

$V^{\mathbf{D}}$: $\mathbf{D} \times \mathbf{V}^{|N|} \longrightarrow \mathcal{V}$;

$(d,V) \longmapsto S^{\mathbf{D}}_{\Pi(d)}(V^{\mathbf{A}}(A_1, V), ..., V^{\mathbf{A}}(A_p, V))$

for $d = (A_1, ..., A_p, c)$

Finally, the value for each class of antecedents is defined to be the sum of the values assigned to each node in the class.

$V^{\mathbf{A}}$: $Pw(\mathbf{N}) \times \mathbf{V}^{|\mathbf{N}|} \longrightarrow \mathcal{V}$;

$(A,V) \longmapsto S^{\mathbf{A}}_{i(A)}(V(a_1), ..., V(a_q))$

for $A = \{ a_1, ..., a_q \}$

The valuation functions provide a framework in which to place the user defined summation functions. Strictly speaking each summation function is a one place function taking an arbitrary n-tuple of values $X = (v_1, ..., v_n) \subseteq \mathcal{V}^*$ and returns a single value $v \in \mathcal{V}$, i.e.

$$S: Pw(\mathcal{V}^*) \longrightarrow \mathcal{V};$$
$$(v_1, ..., v_n) \longmapsto S((v_1, ..., v_n))$$

For convenience sake, to prevent a proliferation of brackets, I have treated them as variable arity functions thus

$$S: \mathcal{V}^* \longrightarrow \mathcal{V};$$
$$(v_1, ..., v_n) \longmapsto S(v_1, ..., v_n)$$

There are no restrictions placed on the summation functions save the following: a node with no attached dependencies shall have a value *nil*, and a node with a single dependency shall have the value of that dependency. I.e.

Defn $V^{\mathbf{N}}(n, V) = S^{\mathbf{N}}(\varnothing) = nil$ where $\mathbf{D}(n) = \varnothing$, and

$V^{\mathbf{N}}(n, V) = S^{\mathbf{N}}(v) = v$ where $\mathbf{D}(n) = \{d\}$ and $V^{\mathbf{D}}(d, V) = v$

The first restriction forces a correspondence between the meaning of *nil* as the value corresponding to "no supported value exists", and the fact that only dependencies are capable of supporting values. The second restriction corresponds to the intuition that the summation functions are a way of combining information about evidence or belief. If there is only a single value, i.e. there is no other support to be considered, then that value should remain unchanged.

We are now in a position to define those valuations in which the values assigned to the nodes correspond to the support given to them by their attached dependencies. We shall call such valuations *admissible*.

Defn A valuation V of **<N,D>** is *admissible* iff $\forall$ n $\in$ **N**, V(n) = $V^{\mathbf{N}}$(n, V)

I shall write VA(**<N,D>**) for the set of admissible valuations of **<N,D>**.

We can also formally define the notion of a dependency supporting the value of a node, given a particular valuation. Intuitively we want d to support n if the value of n is a function of the value of d so that if the value of d is changed then the value of n is changed. Thus

Defn $d_i \in D(n) = \{d_1, ..., d_m\}$ *supports* $V^{\mathbf{N}}(n, V)$ iff $\exists$ v $\in \mathcal{V}$ s.t.

$S^{\mathbf{N}}(V^{\mathbf{D}}(d_1, V), ..., V^{\mathbf{D}}(d_{i-1}, V), v, V^{\mathbf{D}}(d_{i+1}, V), ..., V^{\mathbf{D}}(d_m, V)) \neq$

$S^{\mathbf{N}}(V^{\mathbf{D}}(d_1, V), ..., V^{\mathbf{D}}(d_m, V))$

and v is a value that could actually assigned to some dependency[4].

3.3 Semantics of IDNs

So far, this chapter has covered the need for formal semantics; sketched the informal semantics of support; defined the network and interpretation structures for IDNs; and provided a formal definition of a well supported interpretation, i.e. an admissible valuation. This in itself is sufficient for the work that follows in the rest of the thesis.

However, if we wish to relate the structure of the network to the admissible valuations in such a way as to be able to prove relationships between the values of different nodes we need more structure. I.e. for IDNs to be a formal system with a well defined semantics, equivalent to a logic of some sort, we must have the following (from [Patel-Schneider 1987])

- a syntactic language of sentences $\mathcal{L}$;
- a class of semantic structures $\mathcal{C}$;
- a set of truth values $\mathcal{V}$;
- and an interpretation function i that maps sentences in the language to elements of the set of truth values, given a particular semantic structure. I.e.

$$\forall \alpha \in \mathcal{L}, S \in \mathcal{C}. i(\alpha, S) \in \mathcal{V}$$

[4] Given the presence of assertional dependencies, this should always be the case.

For example, first-order logic consists of a language containing constant and variable symbols, function and predicate symbols, and quantifiers and connectives e.g. $\wedge$, $\vee$, $\forall$. In Tarskian semantics of FOL [Boolos *et al* 1974] the semantic structures consist of:

- a domain or universe of discourse from which objects can be assigned to variables;
- a mapping from constants in the language to objects in the domain (denotation);
- a mapping from functions in the language to functions in the domain; and
- a characteristic function for each predicate, mapping sets of objects in the domain to truth values.

The set of truth values is { "true", "false" }, and the interpretation function is just the set of rules for determining the truth value of a compound wff from its constituent elements.

The preceding section defined a variety of structures that could potentially fill one or more of the roles outlined above. The rest of this section is devoted to exploring one possible way of mapping IDN structures to the elements outlined above.

In attempting to relate IDNs to the constituents of a formal system, the first problem is deciding on the language: what are the things we actually want to talk about? Obviously we need to talk about networks and we have defined above what constitutes such a network. We defined a class of semantic structures or valuations. The semantics of support through valuation and summation functions give us an interpretation function for assigning values of "satisfied" or "unsatisfied" to the network, given a particular valuation:

Defn $i(V, \langle \mathbf{N,D} \rangle) =$ "satisfied" iff $V \in VA(\langle \mathbf{N,D} \rangle)$
= "unsatisfied" otherwise

Additionally, we want to talk about individual nodes or dependencies within the network and the relationships between them. For example, if we have an SL-dependency linking a and b, $a \longrightarrow_{SL} b$, that represents an inference of b is believed from a, we want to be able to write theorems like:

Thm "$a \longrightarrow_{SL} b \in \langle \mathbf{N,D} \rangle$ and a being believed (to be true) entails b being believed (to be true)"

and

Thm "a being believed (to be true) entails a $\longrightarrow_{SL}$ b supports b (to be true)"

Before formally defining entailment, we must consider how to formally write such theorems. In FOL, propositions are either true or false and given the two-valued nature of the semantics one can write " a is false" as $\neg$a and write theorems like:

Thm $\neg b , a \rightarrow b \models_{FOL} \neg a$

The semantics of (FOL) entailment allow a reading of "not b and a implies b entails not a" or "if 'not b' is true and 'a implies b' is true then 'not a' is true" or "if b is false and a implies b then a is false".

Given we may be dealing with three or more values in $\mathcal{V}$, we cannot get away with this (i.e. have prefix modifiers to indicate intended truth values or restrictions on interpretations), but must include the restrictions we want to make in the sentences (this is like introducing meta level predicates). So instead of writing $\models \alpha$ I will write $\models (\alpha$, true). Similarly, instead of $\models \neg\alpha$ I shall write $\models (\alpha$, false). So the basic formulae in our logic of IDNs are pairs consisting of a node or dependency and a value, or extra-logical statements for example about the structure of networks. The nodes/dependencies value pairs are interpreted as follows:

Defn $i(V, (n, v))$ = "true" iff $V(n) = v$
= "false" otherwise

Defn $i(V, (d, v))$ = "true" iff $V^{D}(d) = v$
= "false" otherwise

We can now formalise the theorems above and write

Thm $a \longrightarrow_{SL} b \in \langle \mathbf{N,D} \rangle, (a, in) \models_{IDN} (b, in)$

and

Thm $(a, in) \models_{IDN} (a \longrightarrow_{SL} b, in)$

3.3.1 Semantic Entailment

In general, entailment defines a particular relationship between two formulae over all semantic structures. In particular, for FOL, entailment is defined by:

Defn $\alpha \models_{FOL} \beta$ iff $\forall S \in C, i(S,\alpha) = \text{"true"} \Rightarrow i(S,\beta) = \text{"true"}$

where C is the set of all possible models

and

Defn $\Gamma \models_{FOL} \beta$ iff $\forall S \in C, (\forall \gamma \in \Gamma, i(S,\gamma) = \text{"true"}) \Rightarrow$
$i(S,\beta) = \text{"true"}$

So equivalently for IDNs, entailment relates the "satisfiability" of networks and values for nodes and dependencies. So given formulae that may be networks, dependencies or nodes,

Defn $\alpha \models_{IDN} \beta$ iff $\forall V \in \mathbf{V}^{|\mathbf{N}|}, i(V,\alpha) \in \{ \text{true, satisfied} \} \Rightarrow$
$i(V,\beta) \in \{ \text{true, satisfied} \}$

Defn $\Gamma \models_{IDN} \alpha$ iff $\forall V \in \mathbf{V}^{|\mathbf{N}|}, (\forall \gamma \in \Gamma, i(V,\gamma) \in \{ \text{true, satisfied} \}) \Rightarrow$
$i(V,\alpha) \in \{ \text{true, satisfied} \}$

In using a formal system to represent and reason about knowledge, one attempts to derive conclusions α from a set of initial premises Γ, i.e. one is attempting to establish whether an entailment relation holds, whether $\Gamma \models \alpha$. In using an IDN to represent knowledge, the initial premises are the network used to represent a set of inferences or a theory about some domain. So in most cases, we want to consider entailments between nodes with respect to a given theory, and the network acts as a restriction on the valuations or models considered. The network is effectively a part of the semantic structures - we can talk about assignments to nodes without reference to dependencies but we cannot then define admissible valuations. Even in those cases in which there is no explicit reference to a network, there is an implicit reference. To say $a \longrightarrow_{SL} b$, $(a, in) \models_{IDN} (b, in)$ is equivalent to saying for any admissible valuation of a network containing $a \longrightarrow_{SL} b$, if a is in then b is in.

So I redefine a semantic structure not to be a valuation $V:\mathbf{N} \longrightarrow \mathcal{V}$ but to be a pair $\{\langle \mathbf{N},\mathbf{D} \rangle , V\}$ and the restriction that interpretation of networks are admissible valuations becomes a restriction on the class of semantic structures.

The class of semantic structures is no longer $\mathbf{V}^{|\mathbf{N}|}$ but

$$\mathcal{C} \subseteq (Pw(\mathcal{N}) \times Pw(\mathcal{D})) \times \mathcal{V}^*$$

$$\mathcal{C} = \{\ \{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\} \mid \mathbf{N} \subseteq \mathcal{N}, \mathbf{D} \subseteq \mathcal{C}, \mathrm{V} \in VA(\langle \mathbf{N},\mathbf{D}\rangle)\ \}$$

where $\mathcal{N}$ is an infinite alphabet or set of node names. The interpretation of node/dependency value pairs remains the same, the case of $n \notin \langle \mathbf{N},\mathbf{D}\rangle$ being covered by V(n) = undefined implies $i(\{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\}, (\mathrm{n}, \mathrm{v})) =$ "false". Dependencies are similarly covered, with $i(\{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\}, (\mathrm{d}, \mathrm{v})) =$ "false" if $\mathrm{d} \notin \langle \mathbf{N},\mathbf{D}\rangle$. Additionally, we can now interpret unvalued nodes and dependencies by:

Defn $i(\{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\}, \mathrm{n})) =$ "true" iff $\mathrm{n} \in \mathbf{N}$

= "false" otherwise

Defn $i(\{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\}, \mathrm{d})) =$ "true" iff $\mathrm{d} \in \mathbf{D}$

= "false" otherwise

If we view a network as a conjunction of nodes and dependencies the definition of interpretation of networks becomes:

Defn $i(\{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\}, \langle \mathbf{N}',\mathbf{D}'\rangle)) =$ "true" iff $\mathbf{N}' \subseteq \mathbf{N}, \mathbf{D}' \subseteq \mathbf{D}$

= "false" otherwise

I shall write $\langle \mathbf{N}',\mathbf{D}'\rangle \subseteq \langle \mathbf{N},\mathbf{D}\rangle$ if $\mathbf{N}' \subseteq \mathbf{N}$, $\mathbf{D}' \subseteq \mathbf{D}$. This definition places no semantic conditions on the relationship between $\langle \mathbf{N},\mathbf{D}\rangle$ and $\langle \mathbf{N}',\mathbf{D}'\rangle$. This interpretation of unvalued nodes, dependencies and networks is a way of checking that a given model consisting of the theory represented by $\langle \mathbf{N},\mathbf{D}\rangle$ and its consistent interpretation V actually refers to the objects under consideration.

Our truth values now correspond to {true, false} and the definition of entailment can revert to that of FOL entailment (except for the different semantic structures). In particular, unpacking the definition a little, we get:

Lemma[5] For $\mathrm{n},\mathrm{m} \in \mathbf{N}$, $\mathrm{v},\mathrm{w} \in \mathcal{V}$, $(\mathrm{n}, \mathrm{v}) \models_{\mathrm{IDN}} (\mathrm{m}, \mathrm{w})$ iff

$$\forall\ \{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\} \in \mathcal{C}, \mathrm{V}(\mathrm{n}) = \mathrm{v} \Rightarrow \mathrm{V}(\mathrm{m}) = \mathrm{w}$$

[5] Because of the restriction of network valuation pairs,
$\forall \{\langle \mathbf{N},\mathbf{D}\rangle, \mathrm{V}\} \in \mathcal{C} \equiv \forall \langle \mathbf{N},\mathbf{D}\rangle \in Pw(\mathcal{N}) \times Pw(\mathcal{D}), \forall \mathrm{V} \in VA(\langle \mathbf{N},\mathbf{D}\rangle)$.

Here we can see that the truth values used to talk about nodes and dependencies are not {true, false} but really $\mathcal{V}$ and we can read (n, v) $\models_{IDN}$ (m, w) as "if n is 'v' then m is 'w'" as we would read $\alpha \models_{FOL} \neg\beta$ as "if α is true then β is false".

If we want to talk about entailment in the context of a particular network or networks, we further restrict the domain of semantic structures to include just those networks. So instead of writing

Thm $a \longrightarrow_{SL} b, (a, in) \models_{IDN} (b, in)$

we could write

Thm $(a, in) \models_{T} (b, in)$

where $T = \{ \langle \mathbf{N,D} \rangle \in \mathcal{P}w(\mathcal{N}) \times \mathcal{P}w(\mathcal{D}) \mid a, b \in \mathbf{N},\ a \longrightarrow_{SL} b \in \mathbf{D} \}$

So in order to establish whether a value v assigned to node n entails a value w at node b within the context of a particular theory **<N,D>**, one must consider whether (a, v) $\models_{\langle N,D \rangle}$ (b, w). However the admissible interpretations of **<N,D>** may restrict the valuations considered to such a degree that we do not even consider valuations in which V(a) = v. For example, consider (a, in) $\models_{\langle N,D \rangle}$ (b, in) where $\langle \mathbf{N,D} \rangle = \langle \{a, b\}, \{a \longrightarrow_{SL} b\} \rangle$. The only admissible valuation is V = { (a, out), (b, out)} which means we get the following:

Thm (a, out) $\models_{\langle N,D \rangle}$ (b, out) and (b, out) $\models_{\langle N,D \rangle}$ (a, out)

This problem is the result of missing part of the semantics found in FOL. Part of the Tarskian semantics is the definition of characteristic functions for predicates. This is responsible for asserting the truth or falsity of ground literals and corresponds to an assignment of truth values in a propositional interpretation. Because of the insistence on admissible interpretations we cannot arbitrarily assign values to nodes but must justify those values. It is not enough to consider the network $\langle \{a, b\}, \{\longrightarrow_{SL} a, a \longrightarrow_{SL} b\} \rangle$ for then we get:

Thm (a, in) $\models_{\langle N,D \rangle}$ (b, in) and (b, in) $\models_{\langle N,D \rangle}$ (a, in)

Although this results in the desired behaviour in one case (i.e. (a, in) $\models_{\langle N,D \rangle}$ (b, in)), by only considering the case where the left hand side of the theorem is true, (and therefore the right hand side is true if the theorem is indeed a theorem), it allows the right-hand side (i.e. (b, in)) to prove the left ((a, in)). What is necessary is some way of considering the set of inferences represented by the dependencies in the network to be static and allowing the set of initial

assignments of values to propositions to vary. To this end I define:

Defn An *assertional dependency* is a dependency of the form $\longrightarrow_v$, having no antecedents, and an associated summation function S_v^D such that

$\forall V \in \mathbf{V}^{|N|}$, $V_v^D(V, d) = v$.

Each type of assertional dependency is written $\mathbf{D}_v$ and the set of all assertional dependencies is written $\mathbf{D}_a = \mathbf{D}_{v_1} \cup ,..., \cup \mathbf{D}_{v_n}$.

In general when specifying a particular IDN, one would expect to have an assertional dependency for each node and value combination, the one exception to this being the lack of *nil* value assertions. There are exceptions to this, e.g. see § 3.3.3.

I can now use these dependencies to extend the initial network so as to include a given initial interpretation. This does not preclude the inclusion of assertional dependencies as part of the original network but does include the restriction that not more than one extra assertion is attached to each node. This is merely a restriction of convenience with the action of any two assertions being capable of reproduction by a single assertion. Hence I define the *extensions E of a network*,

Defn $E(\langle \mathbf{N},\mathbf{D}\rangle) = \{ \langle \mathbf{N},\mathbf{D} \cup D\rangle \mid D \subseteq \mathbf{D}_a, \text{ s.t. } \forall n \in \mathbf{N}, |\ D \cap \mathbf{D}(n)| \leq 1 \}$

The definition of entailment is accordingly adjusted.

Defn $\alpha \models_{\langle \mathbf{N},\mathbf{D}\rangle} \beta$ iff $\forall \{\langle \mathbf{N}',\mathbf{D}'\rangle, V\}$,

if $\langle \mathbf{N}',\mathbf{D}'\rangle \in E(\langle \mathbf{N},\mathbf{D}\rangle)$ and $i(\{\langle \mathbf{N}',\mathbf{D}'\rangle, V\},\alpha) =$ "true"

then $i(\{\langle \mathbf{N}',\mathbf{D}'\rangle, V\},\beta) =$ "true"

Returning to the example where $\langle \mathbf{N},\mathbf{D}\rangle = \langle \{a, b\}, \{a \longrightarrow_{SL} b\} \rangle$, the only type of assertional dependency supports a value "in" and corresponds to an SL-justification with no antecedents so I shall write $\longrightarrow_{SL}$ for $\longrightarrow_{in}$ and so $\mathbf{D}_a = \{ \longrightarrow_{SL} a, \longrightarrow_{SL} b \}$.

The semantic structures containing **<N,D>** are

$\{ < \{a, b\}, \{a \longrightarrow_{SL} b\} >, \{(a, \textit{out})\ (b, \textit{out})\} \}$

$\{ < \{a, b\}, \{a \longrightarrow_{SL} b, \longrightarrow_{SL} a\} >, \{(a, \textit{in})\ (b, \textit{in})\} \}$

$\{ < \{a, b\}, \{a \longrightarrow_{SL} b, \longrightarrow_{SL} b\} >, \{(a, \textit{out})\ (b, \textit{in})\} \}$

$\{ < \{a, b\}, \{a \longrightarrow_{SL} b, \longrightarrow_{SL} a, \longrightarrow_{SL} b\} >, \{(a, \textit{in})\ (b, \textit{in})\} \}$

and so we have

Thm $(a, \textit{in}) \models_{<N,D>} (b, \textit{in})$

and

Thm $(b, \textit{out}) \models_{<N,D>} (a, \textit{out})$

The second theorem is akin to the contra-positive, $\alpha \rightarrow \beta \models_{FOL} \neg\beta \rightarrow \neg\alpha$, which appears in a variety of forms. First, if we interpret the negation of formulae by

Defn $i(\{<N,D>, V\}, \neg\alpha)$ = "true" iff $i(\{<N,D>, V\}, \alpha)$ = "false"

= "false" otherwise

then we get the general theorem

Thm If $(a, v) \models_{<N,D>} (b, w)$ then $\neg(b, w) \models_{<N,D>} \neg(a, v)$.

Now, if we interpret the negation of values in node/value pairs by

Defn $i(\{<N,D>, V\}, (n, \neg v))$ = "true" iff $V(n) \neq v$

= "false" otherwise

then we have the following:

Thm If $|\mathcal{V}| = 2$ then $\neg(n, v) \models_{<N,D>} (n, \neg v)$ and $\neg(n, v) \models_{<N,D>} (n, \neg v)$

Notice that in general $\neg(a, v)$ is not equivalent to $(a, \neg v)$. Finally, as a corollary to theorem X, we have the following:

Thm If $|\mathcal{V}| = 2$ and $(a, v) \models_{<N,D>} (b, w)$ then $(b, \neg w) \models_{<N,D>} \neg(a, \neg v)$

3.3.2 Relevant Entailment

One of the main points of this thesis is that dependencies should be used to explicitly represent inferences. If this goal were realised then it should be possible to determine entailment purely by checking the network structure and the corresponding summation functions. Similarly, a node's (n) value should only be entailed by those nodes on which it depends (i.e. *ancs*(n)). There are several problems that prevent this being realised.

The first is the use of standard model theoretic entailment where $\alpha \models \beta$ iff $\mathcal{M}(\alpha) \subseteq \mathcal{M}(\beta)$ where $\mathcal{M}(\pi)$ is the set of models for π (i.e. those semantic structures that satisfy π). This gives the following theorems:

Thm $\perp \models_{\mathcal{M}} \alpha$ where α is any wff and $\perp$ is a contradiction or unsatisfiable wff

and

Thm $\alpha \models_{\mathcal{M}} T$ where α is any wff and T is a tautology or valid wff

In particular, given a set of formulae $\Gamma = \alpha_1, ..., \alpha_n$, if $\Gamma \models_{\mathcal{M}} \beta$ then we can arbitrarily strengthen the antecedent so that $\Gamma \cup \Phi \models_{\mathcal{M}} \beta$ whether or not the formulae in $\Phi = \gamma_1, ..., \gamma_m$ are relevant.

If we want entailment in IDNs to reflect some notion of relevance then the notion of entailment must be restricted or modified to remove this property.

Another problem arises when using a standard notion of negation - i.e. if $\mathcal{M}(\neg\alpha) = \mathcal{C} \setminus \mathcal{M}(\alpha)$. This definition of entailment allows the derivation of contra-positive forms of entailment: if $\alpha \models \beta$ then $\neg\beta \models \neg\alpha$. If entailment is supposed to capture some notion of the nodes in the left hand side being ancestors of nodes in the right hand side, the contra-positive form will pose problems.

Finally, there are problems introduced by having more than two values. Unless the definition of entailment captures some notion of ordering of values it becomes impossible to describe any entailment other than for fixed point values. For example it is impossible to say something along the lines of "whenever α has a belief greater than 0.75, the value of β is greater than 0.9". This issue is explained in more detail in §4.3.2.3.

The upshot of all these problems is that it appears difficult to tie any notion of entailment to the structure of the network.

3.4 Reconstruction of TMSs

Having formally defined what constitutes an Interpreted Dependency Network, I can now reconstruct various TMSs as IDNs. Before doing this I will make a distinction between inferences in the network, and inferences about the network. I will then define summation functions for the J-, A-, and L- TMSs to cover the basic inferences in the network.

3.4.1 Basic and Meta-Level Inferences

In an IDN as in TMSs, the nodes of a network are used to represent propositions and the dependencies in an IDN represent inferences providing support for nodes. These inferences are strictly related to the nodes. As yet there is no way that inferences about the properties or structure of a network can be made or represented **in** that network. Such inferences represent meta-level inferences relating to derivability, unsatisfiability and the like. This lack of meta-level inference in IDNs is at variance with both the spirit of the JTMS, and the extended ATMS [de Kleer, 1986b] where dependencies and other (non-dependency network) structures are used as part of the network in the construction of valuations.

For example, in the JTMS conditional proof (CP) justifications

$$c_{cons}, \{a_1, ..., a_k\}_{in-hyp} \{b_1, ..., b_h\}_{out-hyp} \longrightarrow_{CP} d$$

are valid or *in* "if the consequent node (c_{cons}) is *in* whenever

(a) each node (a_i) of the in-hypotheses ($\{a_1, ..., a_k\}_{in-hyp}$) is *in* and

(b) each node (b_i) of the out-hypotheses ($\{b_1, ..., b_h\}_{out-hyp}$) is *out*."

This is exactly a statement about the entailment of c from $a_1, ..., b_h$ viz, the conditional proof justification is valid if (a_1, *in*) ,..., (a_k, *in*), (b_1, *out*) ,..., (b_h, *out*) $\models_{IDN}$ (c, *in*). In actual fact, as mentioned in §2.1.2, CP-justifications are not evaluated as such, but are dynamically converted to a set of SL-justifications that approximate to the action of the CP-justification.

In the extended ATMS constructs such as *choose*$\{C_1, C_2, \ldots\}$ *and ignore*$\{B\}$ are added. They are meta-level devices because they explicitly deal with the assumptions or values assigned to nodes, i.e. they are constructed from and act directly upon $\mathcal{V}$ which is part of the semantics, and not on **N**. Disjunctions, as represented by *choose*, act as filters on the labels assigned to each node, removing contexts (i.e. sets of assumptions) that cannot contain some assumption C^1 *(because of contradictions), and forcing the inclusion of one assumption from each disjunction in every valuation. Similarly ignore* forces the removal of any environment containing an ignored assumption.

In addition to these structures, operations to change the structure of networks are also considered to be meta-level inferences for they represent changes to the underlying theory or model as represented by the network. In the following sections describing the summation functions for TMSs, I will partition inferences by type, e.g. basic or meta-level, and ignore meta-level inferences. I will return to meta-level inferences in §6.5 where I consider extending IDNs through the use of procedures attached to nodes.

3.4.2 The JTMS as an IDN

For the basic JTMS-IDN we have the following.

- The set of node names $\mathcal{N}$,
- The set of possible dependencies, all support-list justifications,
  $\mathcal{D} = \{ A_{in}, B_{out} \longrightarrow_{SL} c \mid A, B \subseteq \mathcal{N}, c \in \mathcal{N} \} \subseteq Pw(\mathcal{N}) \times Pw(\mathcal{N}) \times \mathcal{N}$
  and the set of assertional dependencies
  $\mathbf{D}_a = \{ \longrightarrow_{SL} a \mid a \in \mathcal{N} \} \subseteq \mathcal{D}$.
- The set of values $\mathcal{V} = \{in, out\}$ with *in* corresponding to "believed to be true" and *out* corresponding to "not believed to be true", equivalent to the *nil* value stipulated by the definition of $\mathcal{V}$ in §4.2.
- The set of summation functions
  $S = \{ S^{N}, S^{D}_{SL}, S^{A}_{SL,in}, S^{A}_{SL,out} \}$[6]

[6] for convenience sake I shall drop the D and A superscripts and write S^{SL} for S^{D}_{SL}, S^{SL}_{in} for $S^{A}_{SL,in}$ and S^{SL}_{out} for $S^{A}_{SL,out}$.

$S^{\mathbf{N}}(\varnothing) = out$

$S^{\mathbf{N}}(v_1, ..., v_n) = out$ iff $\forall i \in [1,n]$, $v_i = out$

$S^{\mathbf{N}}(v_1, ..., v_n) = in$ iff $\exists i \in [1,n]$, $v_i = in$

$S^{\mathrm{SL}}(v_1, v_2) = out$ iff $v_1 = out$ or $v_2 = out$

$S^{\mathrm{SL}}(v_1, v_2) = in$ iff $v_1 = in$ and $v_2 = in$

$S^{\mathrm{SL}}_{\mathrm{in}}(v_1, ..., v_n) = out$ iff $\exists i \in [1,n]$, $v_i = out$

$S^{\mathrm{SL}}_{\mathrm{in}}(v_1, ..., v_n) = in$ iff $\forall i \in [1,n]$, $v_i = in$

$S^{\mathrm{SL}}_{\mathrm{out}}(v_1, ..., v_n) = out$ iff $\exists i \in [1,n]$, $v_i = in$

$S^{\mathrm{SL}}_{\mathrm{out}}(v_1, ..., v_n) = in$ iff $\forall i \in [1,n]$, $v_i = out$

If we replace $\mathcal{V} = \{in, out\}$ with $\mathcal{V} = \{0, 1\}$ then $S^{\mathbf{N}}$ is equivalent to Boolean disjunction, S^{SL} and $S^{\mathrm{SL}}_{\mathrm{in}}$ are Boolean conjunction and $S^{\mathrm{SL}}_{\mathrm{out}}$ is equivalent to Boolean conjunction over a set of negated values.

Of the three basic procedures contained in the JTMS, truth maintenance (the propagation of changes of value) is covered by the task of finding admissible valuations of the networks described above and is the only one to count as basic inferencing. The other two procedures, dependency directed backtracking and the interpretation of CP-justifications through approximations to SL-justifications, are meta-level inferences and are covered in §4.3.2.4.

3.4.2.1 Semantics for the JTMS

Doyle [1983] gives a formal interpretation of the JTMS. Given a set of nodes $\mathcal{N}$ he defines the set of SL- and CP-justifications as

$SL(\mathcal{N}) = Pw(\mathcal{N}) \times Pw(\mathcal{N}) \times \mathcal{N}$

$CP(\mathcal{N}) = SL(\mathcal{N}) \times \mathcal{N}$

and the domain $\mathcal{D}(\mathcal{N}) = \mathcal{N} \cup SL(\mathcal{N}) \cup CP(\mathcal{N})$, otherwise written as $\mathcal{D}$.

The semantics of the JTMS as presented by Doyle is not as clear as it might be, given the classification outlined in §3.3. This is because the domain $\mathcal{D}$ is not only the syntactic language, but also forms the basis of the semantic structures $\mathcal{C}$. However, once the class of

semantic structures has been defined to be the set of *admissible states*, the interpretation of each element of the language is easy.

The truth values consist of the set {*in*, *out*} and given an admissible state $S \in C$ where $S \subseteq D$ (and more besides - $S \subseteq D$ is a necessary but not sufficient condition for membership of C}, the interpretation function i is defined by:

$i(S, d) = in$ iff $d \in S$

$= out$ otherwise

The difficult part of the semantics is in constructing the class of semantic structures for a given domain $\mathcal{D}$. A state S is defined to be a subset of $\mathcal{D}$ and the class of semantic structures is defined by restricting the set of states according to a secondary interpretation function I. This is done by having each component, i.e. $d \in \mathcal{D}$, act "as a restriction on the states in which it may admissibly occur". Individual nodes place no restriction on the states in which they may occur and $I(n) = C$, unless the node represents a contradiction which can occur in no state thus $I(n) = \varnothing$. In this context we can (must) read $n \in S$ as "n is believed in S".

Justifications place more complex restrictions on states. Given the view of CP-justifications as meta-reasoning, I will only consider the interpretation of SL-justifications. So for $d = A_{in}, B_{out} \longrightarrow_{SL} c$, $I(d) = \{ S \subseteq \mathcal{D} \mid A \subseteq S \subseteq \mathcal{D} \wedge B \rightarrow c \in S \}$. In other words I removes those states in which the antecedents of a justification are satisfied but the consequent is not satisfied.

It is harder to give an intuitive reading for $d \in S$ than for nodes. It could be read as "d is currently believed (valid) in S" i.e. is *in* and actually supports c. However, a better reading (given the way interpretations are constructed) would be "d exists in S" i.e. it expresses a relationship between the values of c and the antecedents of d (but may not actually be supporting d in the current state).

Admissible states (denoted by the set S_A) are those sets of components that are self-satisfying i.e. each component is sanctioned by the interpretation of the other components. Formally,

$$S_A = \{\, S \subseteq \mathcal{D} \mid S \in \bigcap_{d \in S} I(d) \,\}$$

According to Doyle, the basic action of the JTMS is modelled by having a set of nodes and justifications S, which may or may not be an admissible state, and extending S to find an admissible extension *E*. The starting state S is a set of nodes and dependencies that are believed to be true, i.e. are *in*. The admissible extensions *AExts(S)* of S are those admissible states containing S (so S is also included in the semantic structures that interpret it!) that are *finitely grounded* (i.e. have no circular supporting structures). These states have no extra justifications added and any additional nodes must be supported by nodes and dependencies in S and/or other nodes which have been added through a series of expansions that do not add circular arguments.

It is possible to construct a (partial) mapping from Doyle's description of the JTMS to the IDN version. At the language or object level, the initial state S maps to a dependency network **<N,D>** with the SL-justifications in S mapping to equivalent SL-dependencies in **D**, and the nodes in S mapping to assertional dependencies in **D**. In this situation, the dependencies in S are equivalent to a network and the set of nodes to a set of premises represented by assertional dependencies:

$$S \equiv < \bigcup_{d \in D} \mathit{nodes}(d) \,,\, D \cup \{\, \longrightarrow_{SL} n \mid n \in N \,\} >$$

where $N = \{\, n \in S \mid n \in \mathcal{N} \,\}$ and $D = \{\, d \in S \mid d \in SL(\mathcal{N}) \,\}$

At the semantic level, Doyle's interpretations of a network, i.e. the admissible extensions *AExts(S)* should map to the admissible valuations *VA*(**<N,D>**). However, because the JTMS excludes self-supporting cycles (i.e. the interpretations are grounded) but the JTMS-IDN does not, we have *AExts(S)* $\subseteq$ *VA*(**<N,D>**). It is necessary to restrict the set of admissible valuations in some way so that *AExts(S)* $\subseteq$ *VA*(**<N,D>**).

Although taking minimal admissible valuations produces the right restrictions for monotonic cycles, in general it is not possible simply to equate admissible extensions with minimal admissible valuations. Although finite-groundedness implies minimality, the converse is not true.

For example, given the dependencies

$$\mathbf{D} = \{a_{out} \longrightarrow_{SL} a,\ b_{out} \longrightarrow_{SL} b,\ a_{in} \longrightarrow_{SL} b,\ b_{in} \longrightarrow_{SL} a\},$$

the interpretation

$$V = \{(a, \mathit{in})\ (b, \mathit{in})\}$$

is the minimal (and only) admissible valuation but it is not finitely grounded.

What can be said is that the admissible valuations of the JTMS-IDN **do** correspond to the notion of *locally grounded* extensions of a network. Given the following:

Defn a state E is locally grounded in S iff

$$\forall x.\ x \in E \Rightarrow x \in S \vee$$
$$\exists r = A_{in}\ B_{out} \longrightarrow_{SL} x.(r \in E \wedge A \subseteq E \wedge B \cap E = \varnothing).$$

and the fact that E is admissible iff

$$\forall r = A_{in}\ B_{out} \longrightarrow_{SL} x.\ r \in E \wedge A \subseteq E \wedge B \cap E = \varnothing \rightarrow x \in E$$

then if S is a set of justifications then a locally grounded admissible extension of S is a state $S' = S \cup E$ s.t.

$$\forall x.\ x \in E \leftrightarrow \exists r = A_{in}\ B_{out} \longrightarrow_{SL} x.\ \ r. \in S \wedge A \subseteq E \wedge B \cap E = \varnothing.$$

In other words, the nodes that are in the extension (admissible valuation V) are exactly those that have a valid justification in E: ($V^{\mathbf{N}}(n, V) = V^{\mathbf{D}}(d_i, V) = \mathit{in}$ for some i).

In order to get a correspondence between minimality and groundedness it is necessary to consider *stable set* semantics [Elkan 1990] where $V \in VA(\langle \mathbf{N}, \mathbf{D} \rangle)$ is finitely grounded iff it is the minimal admissible valuation of $\langle \mathbf{N}, \mathbf{D}_V \rangle$ where

$$\mathbf{D}_V = \{\ A_{in} \longrightarrow_{SL} c \mid \exists d' \in D.\ d' = A_{in} B_{out} \longrightarrow_{SL} c$$
$$\wedge\ \forall a \in A.\ V(a) = \mathit{in} \wedge \forall b \in B.\ V(b) = \mathit{out}\ \}.$$

Whether self-satisfying cycles should be allowed is at this time a moot point[7]. The important point is that semantically speaking, the interpretation of networks in the JTMS-IDN is different to those in the original JTMS. One possible solution to this is to modify the

[7] Given the discussion in §4.2.1 about the psychological plausibility of circular justifications, e.g. "I believe P because I believe P", it seems for some applications finite groundedness is too strong a condition.

interpretation mechanism for cycles so that there is built-in indication of groundedness. This approach will be outlined later in §4.2.6, but for the moment we must be satisfied with agreement between the "real" JTMS and the IDN version on the interpretation of acyclic networks.

3.4.3 The ATMS as an IDN

For the basic ATMS-IDN we have the following.

- The set of node names $\mathcal{N}$ = including a distinguished node $n_\perp$ used to represent contradictions.
- The set of possible dependencies, all of a single type arbitrarily called support dependencies or s-type for short[8]

  $\mathcal{D} = \{ A_{+ve}, n_{\perp_w} \longrightarrow_s c \,/\, A \subseteq \mathcal{N}, c \in \mathcal{N} \} \subseteq Pw(\mathcal{N}) \times \{n_\perp\} \times \mathcal{N}$

  and the set of assertional (a-type) dependencies:

  $\mathbf{D}_a = \{ \longrightarrow_v n \mid n \in \mathcal{N}, prop(n) = A, v = \{A\} \subseteq \mathcal{V} \text{ for } A \in \mathcal{A} \}$
- The set of values $\mathcal{V} = Pw(Pw(\mathcal{A}))$ for some distinguished set of *assumptions* $\mathcal{A}$ with $\{\}$ corresponding to the *nil* value.
- The set of summation functions $\boldsymbol{S} = \{ S^N, S^D_s, S^A_{s,+ve}, S^A_{s,-ve} \} \cup \{ S^D_v \mid v = \{A\} \in \mathcal{A} \}$

$$S^N(\varnothing) = \{\}$$

$$S^N(v_1, \ldots, v_n) = reduce(\bigcup_{i \in [1,n]} v_i)$$

$$S^D_s(v_1, v_2) = remove(v_1, v_2)$$

$$S^A_{s,+ve}(v_1, \ldots, v_n) = reduce(\{ v = \bigcup_{1 \le i} x_i \mid x_1, \ldots, x_n \in v_1 \times \ldots \times v_n \})$$

$$S^A_{s,-ve}(v) = v$$

[8] The structure of the dependencies have been syntactically altered for various reasons discussed after these definitions.

$$S_v^D(\varnothing) = \{v\}$$

where the functions *remove* and *reduce* are defined as follows

reduce(v) = $\{e_i \in v \mid \forall e_j, e_j \not\subset e_i\}$ where the label $v = \{ e_1, ..., e_n \}$ with each $e_i \subseteq \mathcal{A}$ being a particular context or set of assumptions.

remove(v_1, v_2) = $\{e_i \in v_1 \mid \forall f_j \in v_2, f_j \not\subset e_i\}$

reduce removes all those environments of v that are subsumed by some other environment thus producing a set of minimal environments.

remove removes those environments of v_1 that are supersets of environments in v_2. Given the construction of the dependency, v_2 will be the *no-goods* or contradictory sets of assumptions, so the application of the function removes all contradictory environments from v_1.

As noted above (in the footnote below!), the structure of the dependencies in the ATMS-IDN are somewhat different to those in the ATMS, but this does not change the semantics nor the operation of the system.

The change in dependency structure is made so that the ATMS-IDN conforms to the idea of local computation of values. A new antecedent is included in all s-type dependencies to convey the no-goods, i.e. the environments contained in the label of the contradiction node $n_\perp$, each of which must not be contained in any environment of another node's label. This makes the s-type ATMS dependency equivalent to a restricted JTMS SL-dependency. It is restricted in that the only allowed negative antecedent, i.e. nodes that can place restrictions on what must be disbelieved, is the contradiction node n-0.

Another difference between the ATMS and the ATMS-IDN is the inclusion of assertional dependencies in the ATMS-IDN. In the ATMS, assertional dependencies do not exist as they do with the JTMS. Nodes are simply designated as assumptions and given a corresponding label. The assumption nodes, representing statements like "assume the sky is blue", are then used to justify assumed nodes representing the actual assumed datum, "the sky is blue". In general, the assumption nodes may be justified by other nodes [de Kleer

1986a, p147] but de Kleer avoids doing this in the paper because of the controversy involved [*ibid*, p142], but advocates the use of assumption/assumed node pairs. This then corresponds to the input model of unsupported valuations over root nodes in §3.3.3.3.

Even having chosen to use assertional dependencies, the use of assertional dependencies to model de Kleer's use of assumptions represents a restriction compared to the class of assertional dependencies as originally envisaged in §3.3.1. They only exist for values containing a single environment of one assumption. Furthermore, they should only be attached to the nodes that represent the asserted assumption in the network. Thus, if A = "assume the sky is blue" and *node*(A) = n-11 say, then $\longrightarrow_{A}$n-11 would be the only assertional dependency of type {A}, i.e. supporting the value {A}. Therefore, having chosen our assumptions $\mathcal{A}$, thus defining our values $\mathcal{V}$ and assertional dependencies $\mathbf{D_a}$, a-type dependencies should not be treated as assertional dependencies for the sake of constructing extensions of a network, and E(<**N**,**D**>) = {<**N**,**D**>}. This gives a new set of network dependencies $\mathbf{D}' = \mathbf{D} \cup \mathbf{D_a}$ and assertional dependencies $\mathbf{D_a}' = \varnothing$. There is another way of interpreting assertional dependencies which I shall return to below and which justifies the use of the full set of assertional dependencies.

3.4.3.1 Semantics for the ATMS

In the JTMS the intuitive reading or semantics of the values $\mathcal{V}$ is clear and absolute. V(n) = *in* means that the proposition *prop*(n) associated with n is believed and V(n) = *out* means that it is not believed. Valuations are truth assignments over a network.

For the ATMS the values are not absolute but relative to a particular network and do not give a simple truth assignment to propositions. When designing a specific ATMS-IDN network the designer must specify the distinguished set of assumptions $\mathcal{A}$ and the a-type dependency associated with that assumption. This gives the assumed nodes a fixed value and the belief or truth of other propositions or nodes is defined relative to them. So the values correspond to a set of *possible worlds* indexed by $\mathcal{A}$ where each possible world consists of a different interpretation of the same set of nodes.

Having constructed a valuation consisting of a set of consistent possible worlds, that valuation can then be used to create a particular truth assignment corresponding to a single possible world. This is done by picking a set of assumptions M corresponding to a particular model and checking each node to see if it is true in that particular world. This is the secondary interpretation phase described in §2.1.2.

Although the summation functions above would seem to correspond with the informal algorithm given by de Kleer [1986a, p50-152], comparing the possible admissible valuations over cycles it can be seen that the ATMS-IDN does not produce the results sanctioned by the formal semantics [Reiter and de Kleer 1987, ECAI-TMS workshop]. In particular, given $\mathbf{D} = \{a \longrightarrow b, b \longrightarrow a\}$ one should get a valuation $V = \{(a\ \{\})\ (b\ \{\})\}$ but any valuation $V' = \{(a\ L)\ (b\ L)\}$ will be admissible for the ATMS-IDN. To remove this anomaly it is sufficient to choose the minimal admissible valuation[9].

The choice of minimal admissible valuation for the correct interpretation of ATMS networks introduces another problem: the derivation and recording of contradictions. The process of dealing with contradictions has two parts. First contradictions are generated by supporting $\perp$ with some dependency $A \longrightarrow \perp$. The resulting label of $\perp$ are the no-good or inconsistent environments. Secondly, the no-goods are removed from each nodes label, thus preventing them from being derived under contradictory circumstances. However, this may result in the no-good being removed from $\perp$.

For example, consider $\mathbf{D} = \{ \longrightarrow_{\{\{A\}\}} a,\ a \longrightarrow b,\ b \longrightarrow \perp \}$. If V is an admissible valuation then $V(a) = \{\{A\}\}$, $V(b) = \mathit{remove}(\{\{A\}\}, V(\perp))$ and $V(\perp) = V(b)$. I.e. b's label is $\{\{A\}\}$ with any inconsistent environments removed, while the inconsistent environments are exactly those environments of b that are consistent - $\{A\}$ is no-good iff it is not no-good. It is only when an environment first appears in the justification supporting $\perp$ that this loop is avoided.

There are several solutions to the problem of recording contradictions: either allow a self-supporting justification of no-goods and let admissible valuations be non-minimal over

[9] This works for the ATMS and not the JTMS because of the monotonic nature of the cycles which means that minimality implies groundedness.

$\perp$; or use justifications to directly record contradictions. E.g. if {A B C} is determined to be no-good, say from d$\longrightarrow\perp$ and V(d) = {{A B C}}, then add {A B C} $\longrightarrow \perp$ to the network. This problem shows the difficulty of trying to treat a meta-level concept (e.g. contradictions) within the object framework. In both cases it is difficult to maintain the origin of the contradiction but remove its affect from the network.

It is obvious why the second is desired (maintaining a contradiction-free labelling is the purpose of the ATMS) but the first is equally necessary if changes to network are to be allowed. Not only may contradictions be explicitly removed by removing dependencies supporting $\perp$, but they may also be removed implicitly by removing some of the other intermediate dependencies linking the assumptions to $\perp$. Given it is not possible to maintain constant support for no-goods it becomes necessary to explicitly check the validity of no-goods after the removal of any dependencies. For this reason the removal of justifications can be computationally expensive[10].

What is harder to capture is the notion of entailment. As described above, $\mathbf{D}_a = \varnothing$ and so $E(\langle \mathbf{N},\mathbf{D}\rangle) = \{\langle \mathbf{N},\mathbf{D}\rangle\}$. Given there is only a single admissible valuation for any ATMS-IDN network, the following theorem results:

Thm $\Gamma \,{}_{R}\!\models_{\langle \mathbf{N},\mathbf{D}\rangle} \alpha$ iff $\Gamma = \varnothing$ and $\alpha \in \{ (n, v) \mid n \in \mathbf{N}, v = V(n), V \in VA(\langle \mathbf{N},\mathbf{D}\rangle)\}$

Proof Easy

We have returned to the problem of §3.3.1 of only having a single extension which we overcame by introducing assertional dependencies and which has now (unsurprisingly) reappeared when $\mathbf{D}_a = \varnothing$.

[10] De Kleer suggests not actually removing justifications but adding an extra antecedent assumption to each dependency to represent its validity. To retract the dependency this assumption is simply removed from the problem solving context. However, given the complexity of calculating labels is exponential in the size of environments [Provan 1990] this will quickly lead to a combinatorial explosion given a network of some depth and interconnectedness.

3.4.3.2 Assumptions and Assertional Dependencies

The initial restriction of $\mathbf{D}_a$ to the empty set was based on the premise that assertional dependencies in the ATMS-IDN represented specific assumptions, and should only be attached to those nodes that represented those assumptions within the network, and that it would be nonsensical to attach them elsewhere. To justify some node m, other than n = *node*(A), with $\longrightarrow_A$ de Kleer intimates that it would be better to use $n_{+ve}\longrightarrow_s m$.

I would argue that a case could be made for extending $\mathbf{D}_a$. The set of assumptions $\mathcal{A}$ has no external semantic content. They are merely arbitrary devices for tracing derivations of nodes. Though de Kleer would have us use the same token for a node (or proposition attached to a node) and a semantic value, there is no need for this connection. One can simply define the meaning of an assumption $\in \mathcal{A}$ as the conjunction over the propositions related to the nodes to which it is attached. For example, if the dependency $\longrightarrow_A$ were attached to nodes n and m with respective propositions π and ψ, the assumption A would read π and ψ are true.

The argument that no-goods are defined by particular sets of assumptions and thus would change their meaning if assumptions were arbitrarily assigned meaning does not hold water. The inconsistency represented by a no-good should be between the nodes in the network and therefore defined by the nodes' current labels, and not directly defined through assumptions. For example, to say n contradicts m, the dependency $\{n, m\}_{+ve}\longrightarrow_s n_\perp$ should be added to the network - in this way a record of the contradiction is kept in the network. If the intersection of the labels of n and m were declared no-good by some reasoning or procedure outside of the network interpretation there would be no record in the structure of the network, defeating the purpose of a foundations approach to belief revision.

So, given $A \in \mathcal{A}$ has no particular semantic import but is defined relative to the nodes to which $\longrightarrow_{\{A\}}$ is attached, we can let

$$\mathbf{D}_a = \{ \longrightarrow_v n \mid n \in \mathcal{N}, v = \{A\} \text{ for } A \in \mathcal{A}\}.$$

Given that assumptions represent tags for tracing derivations it is still useful to restrict assertional dependency types to singleton values.

The extensions of a network $E(\langle \mathbf{N},\mathbf{D}\rangle) = \{ \langle \mathbf{N},\mathbf{D}\cup D\rangle \mid D \subseteq \mathbf{D_a} \}$ represent different views of a given theory. The assumptions $\mathcal{A} = \{cons(d) \mid d \in D\}$ designate the starting points of the variable area of the theory represented by the network. Any node not containing an assumption in its set of ancestors will either always be true or always be false depending on whether its label is $\{\{\}\}$ or $\{\}$.

By changing where the assumptions are attached in the network, we get a different focus on the theory. In other words, rather than judging belief in a node relative to one set of assumptions, belief is now relative to a second set.

This change in focus can represent a simplification, ignoring part of the theory. For example, given the network

$$\langle \mathbf{N},\mathbf{D}\rangle = \langle \{n,m,l\}, \{n,m \longrightarrow_s l\} \rangle$$

and a set of assumptions represented by

$$D = \{\longrightarrow_A n, \longrightarrow_B m\}$$

then $V(l) = \{\{A, B\}\}$ for $V \in VA(\langle \mathbf{N},\mathbf{D}\cup D\rangle)$ - l is believed if A and B are believed. If the set of assumptions is changed:

$$D = \{\longrightarrow_C l\}$$

then $V(l) = \{\{C\}\}$ - l is believed only if C is believed. The second interpretation of the network is simpler in that it has ignored n and m and concentrated on the network at and below l.

By replacing $\{\longrightarrow_A n, \longrightarrow_B m\}$ by $\{\longrightarrow_C n, \longrightarrow_C m\}$ the level of focus has remained the same, i.e. nodes below n and m would be variable with regard to C, but the granularity has decreased - the assumption C represents the conjunction of A and B.

Similarly, if we add an assumption where there was an empty label this corresponds to a more detailed theory (with a wider area of variability) and we are increasing the granularity at which things are interpreted. There are more distinct possible worlds to choose from when selecting just a single interpretation within which to work.

Having re-defined $\mathbf{D_a}$, each network is now capable of supporting a variety of models as it is possible to extend the network by the addition of a number of assertional dependencies. The problems that arise by having a single interpretation of a network are alleviated.

3.4.4 The LTMS as an IDN

To model the LTMS we have to introduce a few changes that alter the form of the system, though not the semantics. Firstly, though the LTMS is three-valued, the easiest way to record contradictions is to have an additional fourth value. Secondly, in the LTMS propositional formulae are converted to a set of constraints which under certain circumstances are used to derive values for nodes. These constraints are further converted to a set of dependencies as outlined below.

For the basic LTMS-IDN we have the following.

- The set of node names $\mathcal{N}$
- The set of dependencies which are of two types, those providing positive support and those providing negative support,

  $\mathcal{D} = \{ A_t, B_f \longrightarrow_v c \mid A, B \subseteq \mathcal{N}, c \in \mathcal{N}, v \in \{t, f\} \} \subseteq Pw(\mathcal{N}) \times Pw(\mathcal{N}) \times \mathcal{N}$

  and the set of assertional dependencies

  $\mathcal{D}_a = \{ \longrightarrow_v a \mid a \in \mathcal{N}, v \in \{t, f\} \} \subseteq \mathcal{D}$
- The set of values $\mathcal{V} = \{t, f, n, c\}$ with t corresponding to "believed to be true" and f corresponding to "believed to be false", c corresponding to "contradictory" or believed to be both true and false, and n corresponding to "no belief either way" equivalent to the *nil* value stipulated by the definition.
- The set of summation functions $S = \{S^N, S_t^D, S_{t,t}^A, S_{t,f}^A, S_f^D, S_{f,t}^A, S_{f,f}^A\}$

  though I will write S_t^A for $S_{t,t}^A$ and $S_{f,t}^A$ and S_f^A for $S_{t,f}^A$ and $S_{f,f}^A$ as the functions are the same.

$S^N(\varnothing) = n$

$S^N(v_1, ..., v_n) = n$ iff $\forall i \in [1,n], v_i = n$

$S^N(v_1, ..., v_n) = t$ iff $\exists i \in [1,n], v_i = t$ and $\forall i \in [1,n], v_i \neq f$

$S^N(v_1, ..., v_n) = f$ iff $\exists i \in [1,n], v_i = f$ and $\forall i \in [1,n], v_i \neq t$

$S^N(v_1, ..., v_n) = c$ iff $\exists i,j \in [1,n], v_i = t$ and $v_j = f$

$S_t^D(v_1,v_2) = n$ iff $v_1 = f$ or $v_2 = f$

$S_t^D(v_1,v_2) = t$ iff $v_1 = t$ and $v_2 = t$

$S_f^D(v_1,v_2) = n$ iff $v_1 = f$ or $v_2 = f$

$S_f^D(v_1,v_2) = f$ iff $v_1 = t$ and $v_2 = t$

$S_t^A(v_1, ..., v_n) = n$ iff $\exists i \in [1,n], v_i \neq t$

$S_t^A(v_1, ..., v_n) = t$ iff $\forall i \in [1,n], v_i = t$

$S_f^A(v_1, ..., v_n) = n$ iff $\exists i \in [1,n], v_i \neq f$

$S_f^A(v_1, ..., v_n) = t$ iff $\forall i \in [1,n], v_i = f$

As with the JTMS-IDN I will not consider backtracking at this point as it represents a meta-level activity. However, it is worth noting that McAllester allows the extension in the type of assertional dependencies allowed so that rather than simply having premises that are positively supported, one can differentiate between the strengths or degree of belief in the assertions. This enables the backtracking algorithm to retract weaker assumptions before stronger ones.

3.4.4.1 Semantics for the LTMS

Semantically the LTMS is a restricted JTMS and the former can be transformed into the latter as follows. Each proposition p in the LTMS is represented by two in the JTMS, say t(p) corresponding to "p is true" and f(p) to "p is false" and values in the LTMS can be mapped to valuations over these two nodes. Given *node*(p) = n, *node*(t(p)) = m and *node*(f(p)) = l,

$V(n) = n \equiv V(m) = out$ and $V(l) = out$

$V(n) = t \equiv V(m) = in$ and $V(l) = out$

$V(n) = f \equiv V(m) = out$ and $V(l) = in$

$V(n) = c \equiv V(m) = in$ and $V(l) = in$

Dependencies in the LTMS can be correspondingly translated into the JTMS:

$A_t, B_f \longrightarrow_t c \equiv A'_{in} \longrightarrow_{SL} t(c)$ and $A_t, B_f \longrightarrow_f c \equiv A'_{in} \longrightarrow_{SL} f(c)$

where $A_t = \{a_1, ..., a_n\}$, $B_t = \{b_1, ..., b_n\}$

and $A'_t = \{t(a_1), ..., t(a_n), f(b_1), ..., f(b_n)\}$

It is now possible to see why the LTMS is a restriction of the JTMS, no nodes will appear in the outlist B_{out} of any dependency. This means that the LTMS is not capable of non-monotonic reasoning through the use of explicit references to missing information.

3.4.4.2 Propositions, Constraints and Dependencies

Given the LTMS is effectively a restriction on the JTMS, the most interesting aspect of it is the coding of propositions as constraints in the LTMS which are then turned into dependencies in the LTMS-IDN.

McAllester translates the standard connective into constraints on the values assigned to propositions in compound formulae or correlations between their values. This can be done by taking the definition of what a literal proposition means and turning it into conjunctive normal form. Thus the proposition "a and b" should be equivalent to the logical formula $a \wedge b$. Thus:

$$\text{"a and b"} \leftrightarrow a \wedge b \equiv a \wedge b \rightarrow \text{"a and b"} \equiv \text{"a and b"} \vee \neg a \vee \neg b$$
$$\equiv \text{"a and b"} \rightarrow a \equiv (\neg\text{"a and b"} \vee a)$$
$$\equiv \text{"a and b"} \rightarrow b \equiv (\neg\text{"a and b"} \vee b)$$

which McAllester translates into a set of constraints in which at least one node/value pair must be satisfied in all states of the LTMS.

(("a and b", *t*) ("a", *f*) ("b", *f*))

(("a and b", *f*) ("a", *t*))

(("a and b", *f*) ("b", *t*))

The basic action of the LTMS is to derive values for propositions having nil values when that assignment is the only possible way of satisfying an existing constraint. Thus in the example above, if the value of "a and b" = *f*, the value of "a" = *t* and the value of "b" = *nil* then the LTMS would assign the value *f* to "b" in order to satisfy the first constraint.

In general each disjunctive constraint can be translated into a set of implications to capture this process of assignment, and these implications can in turn be represented by dependencies. Given a constraint with a set of positive formulae $P = \{p_1, ..., p_n\}$, i.e. formulae that satisfy the constraint if true, and a set of negative formulae $Q = \{q_1, ..., q_m\}$, i.e. formulae that satisfy the constraint if false, then

$$((p_1, t), ..., (p_n, t)\ (q_1, f), ..., (q_m, f)) \equiv$$
$$\{ Q_t, P'_f \longrightarrow_t p_i \mid P' = P \setminus \{p_i\}, i \in [1,n]\} \cup$$
$$\{ Q'_t, P_f \longrightarrow_f q_j \mid Q' = Q \setminus \{q_j\}, j \in [1,m]\}$$

For example

(("a and b", *t*) ("a", *f*) ("b", *f*)) ≡

{"a and b"$_f$, "a"$_t \longrightarrow_f$ "b", "a and b"$_f$, "b"$_t \longrightarrow_f$ "a", "a"$_t$, "b"$_t \longrightarrow_t$ *"a and b"*

(("a and b", f) ("a", *t*)) ≡ {"a and b"$_t \longrightarrow_t$ "a" , "a"$_f \longrightarrow_f$ "a and b"}

(("a and b", *f*) ("b", *t*)) ≡ {"a and b"$_t \longrightarrow_t$ "b" , "b"$_f \longrightarrow_f$ "a and b"}

I define *and*(x, y, z) to be a function that returns exactly this set of dependencies with every occurrence of "a and b", "a" and "b" replaced by x, y, and z respectively.

We can use this method to derive patterns of dependencies to model the structure of propositions that can be expressed in terms of subformulae and the connectives $\wedge$, $\vee$, $\neg$ and $\rightarrow$. This can be done in any IDN that has a dependency type indicative of positive support (confirming evidence), and that has (or allows) nodes for a proposition and its negation. More importantly, we can show that the semantics of a set of dependencies encoding a logical connective corresponds to the semantics of that connective. E.g.

Thm $\forall \{<\mathbf{N},\mathbf{D}>, V\} \in \mathcal{C}$ s.t. *and*(n, m, l) $\subseteq \mathbf{D}$,
V(n) = t iff V(m) = t and V(l) = t.

3.4.5 Contradictions

TMSs are mechanisms to handle contradictions, but until this point there has been little mention of them in the reconstruction of the J-, A-, and L- TMSs. Before looking at the different ways to represent them, I want to consider how they occur and what happens when they do occur in normal logical theories, and in problem solving.

Formally, a contradiction is a proposition of the form "$\alpha \vee \neg\alpha$" though informally, it can be used, say in the context of a problem solver or some theory, to mean it is not possible that two particular propositions are true at the same time [Collins Dictionary]. The second case is an abstraction of the first though ultimately it leads to the first case. For example, α may contradict γ given $\alpha \rightarrow \beta$ and $\beta \rightarrow \neg\gamma$.

Similarly, there are two uses for contradictions which are closely related. The first is to exclude interpretations of a set of facts or theories that would derive a contradiction. So given the example above, one would not consider models of the theory $\{\alpha \rightarrow \beta$ and $\beta \rightarrow \neg\gamma\}$ in which α and γ were both true. Alternatively, if one were trying to construct a theory to model a particular set of known situations, the derivation of an inconsistency should lead to the abandonment of (parts of) the theory. More specifically, one uses contradictions to construct or prove theorems using *reductio ad absurdum*. Here the derivation of a contradiction is used to prove the negation of one of the axioms used in the proof[11].

This distinction is reflected in the way the various TMSs represent and deal with contradictions. The ATMS uses them to exclude particular interpretations by labelling assumption sets as *no-good* and preventing these sets from being simultaneously believed. Both the J- and L-TMSs use dependency directed backtracking to retract, either automatically or via the user, one of the assumptions underlying the contradiction. Again, the LTMS uses a strict definition of a contradiction, being the simultaneous labelling of a node

[11] Of course this method of proof is not acceptable to all e.g. intuitionist logicians and mathematicians.

as both *t*(rue) and *f*(alse), whereas the JTMS uses the broader definition whereby any node can be designated a contradiction node, that contradiction occurring when the node is labelled *in*.

The way the three TMSs are implemented as IDNs demonstrates the various ways and levels at which contradictions can be defined. The JTMS-IDN operates both at the language level and the meta-level. Particular nodes are designated contradictions, which are different from normal nodes in that DDB is invoked when they are believed, thus giving a procedural semantics to contradictions which can also be viewed as meta-level objects as they change the underlying structure of the network. The LTMS-IDN operates at the interpretation level by having a particular value to represent contradictions, this value being assignable to any node. The ATMS similarly works at this level not through explicit reference, but by the removal of particular interpretations.

3.5 Gardenfors' Postulates, the JTMS and IDNs

In Chapter 1 I presented Gardenfors' postulates for belief revision and took issue with the coherence approach to belief revision that they seemed to support. In Chapter 2 I argued that TMSs provided the kind of functionality desirable for belief revision devices without reference to Gardenfors' belief revision postulates (GBRP). Gardenfors [1988, p33-35] also considers Doyle's TMS as a belief revision system but fails to analyse how the JTMS relates to GBRP. In this section I shall define what constitutes a belief state for an IDN in general and show how the JTMS fails to satisfy some of GBRP in particular, and what conditions are necessary for an IDN to satisfy GBRP.

As shall be argued in §5.0.1, admissible valuations are analogous to closed consistent interpretations of a theory represented by a given network. Therefore the set of belief states can simply be identified with the set of semantic structures:

$$\mathcal{C} = \{ \{\langle \mathbf{N},\mathbf{D}\rangle, \mathbf{V}\} \mid \mathbf{N} \subseteq \mathcal{N}, \mathbf{D} \subseteq \mathcal{C}, \mathbf{V} \in VA(\langle \mathbf{N},\mathbf{D}\rangle) \}.$$

Gardenfors also requires a subsumption test or ordering for belief states (e.g K^+3, K^+5, K^+6, K^-2, K^-5) which for simple belief states is the subset ordering. Given an ordering $\leq$ on $\mathcal{V}$ an ordering on $\mathcal{C}$ is defined with $\langle\langle \mathbf{N},\mathbf{D}\rangle, \mathbf{V}\rangle \subseteq \langle\langle \mathbf{N}',\mathbf{D}'\rangle, \mathbf{V}'\rangle$ iff $\langle \mathbf{N},\mathbf{D}\rangle$

$\subseteq$ <**N′,D′**> and $\forall n \in \mathbf{N}$. $V(n) \leq V'(n)$. It should be noted that in order to capture the spirit of GBRP the ordering on $\mathcal{V}$ should be a reflection of the degree of truth or belief in a proposition. Thus for the JTMS *in* $\leq$ *out*, while for the ATMS

$\{E_1, ..., E_n\} \leq \{F_1, ..., F_m\}$ iff $\forall E_i \exists F_j . F_j \subseteq E_i$).

The admissible valuation $V \in VA(\langle \mathbf{N,D} \rangle)$ is a reflection of the structure of the theory represented by <**N,D**>, and also reflects the fact that changes in V can only be produced by changes in <**N,D**>. Therefore epistemic inputs can only be made in terms of changes to the network except in the case of ambiguous networks. In the case of ambiguous networks, the construction of admissible valuations is governed by some policy determining which out of a possible set of admissible valuations should be chosen. This in turn defines a *selection function*, $Sel: Pw(C) \longrightarrow C$ (cf [Gardenfors 1988, p77]) on belief states so that a single state is selected from a set of possible states. Given a belief state $K = \langle\langle \mathbf{N,D} \rangle, V \rangle$ the expansion of K by dependency d is defined by

$$K_d^+ = Sel\ (\ \{\langle\langle \mathbf{N,D} \cup \{d\} \rangle, V' \rangle \mid V' \in VA(\langle \mathbf{N,D} \cup \{d\} \rangle)\}\ )$$

Given this definition of expansion the postulates can be tested.

(K^+1) K_d^+ is a belief set:

JTMS - K_d^+ is not necessarily a belief set - if $VA(\langle \mathbf{N,D} \cup \{d\} \rangle) = \varnothing$ then <**N,D** $\cup$ {d}> is uninterpretable. The classic example is an odd loop:

$K = \langle \langle \{a, b\}, \{a_{out} \longrightarrow_{SL} b\} \rangle, \{(a, out)\ (b, in)\} \rangle$, $d = b_{in} \longrightarrow_{SL} a$

$VA(\langle \{a, b\}, \{a_{out} \longrightarrow_{SL} b,\ b_{in} \longrightarrow_{SL} a\} \rangle) = \varnothing$

IDN - This is guaranteed only if there are no uninterpretable networks, e.g. the ATMS.

(K^+2) $d \in K_d^+$:

JTMS/IDN - satisfied

(K^+3) $K \subseteq K_d^+$:

JTMS - This is a monotonicity condition on dependencies and given the JTMS contains non-monotonic justifications this is obviously false. E.g. for

$K = \langle\langle \mathbf{N,D} \rangle, V \rangle$ with $\langle \mathbf{N,D} \rangle = \langle \{a, b\}, \{a_{out} \longrightarrow_{SL} b\} \rangle$, $V = \{(a, out)\ (b, in)\}$,

$K_d^+ = \langle\langle \mathbf{N',D'} \rangle, V' \rangle$ where $\langle \mathbf{N',D'} \rangle = \langle \{a, b\}, \{\longrightarrow_{SL} a,\ a_{out} \longrightarrow_{SL} b\} \rangle$ and

$V' = \{(a, in)\ (b, out)\}$ so $K \not\subseteq K_d^+$.

IDN - in general this condition holds only where every summation function is monotonic. Therefore a monotonic JTMS satisfies this postulate but the ATMS does not as an increase in the number of no-goods (through the addition of a new dependency) potentially decreases the node labels as supersets of the no-good are removed.

(K^+4) $\alpha \in K \Rightarrow K_\alpha^+ = K$:

JTMS/IDN - satisfied, as long as K_d^+ is a belief state.

(K^+5) $K \subseteq H \Rightarrow K_\alpha^+ \subseteq H_\alpha^+$;

JTMS - This is another monotonicity constraint and as before it fails for a non-monotonic JTMS: if

$$K = << \{a, b\}, \varnothing >, \{(a, out), (b, out)\} >$$

$$H = << \{a, b\}, \{\longrightarrow_{SL} a\} >, \{(a, in), (b, out)\} >$$

$$d = a_{out} \longrightarrow_{SL} b$$

then

$$K_d^+ = << \{a, b\}, \{a_{out} \longrightarrow_{SL} b\} >, \{(a, out), (b, in)\} >$$

$$H_d^+ = << \{a, b\}, \{\longrightarrow_{SL} a,\ a_{out} \longrightarrow_{SL} b\} >, \{(a, in), (b, out)\} >$$

and $K \subseteq H$ but $K_\alpha^+ \not\subseteq H_\alpha^+$

IDN - Although the update does not have to be monotonic in order to satisfy K^+5, i.e. K_d^+ could be less than K, it does have to be order preserving i.e. K_d^+ is less than H_d^+.

(K^+6) K_α^+ is minimal:

JTMS - this is satisfied by the JTMS but not by the JTMS-IDN unless it operates a policy of selecting minimal interpretations over cycles.

IDN - As with the JTMS-IDN, K^+6 is satisfied iff the selection function picks minimal admissible valuations given a set of admissible valuations. If an IDN is guaranteed to produce a single admissible valuation for any network then the selection function is unnecessary and K^+6 is again satisfied.

Contractions are performed by retracting justifications, either under the control of the user or automatic revision procedures. As with expansions, there may be more than one admissible valuation of the contracted network and a selection function is necessary to determine what the contracted belief state is.

(K^-1) K_d^- is a belief set:

JTMS - this is not guaranteed as the removal of a dependency may result in an uninterpretable network: $K = <<\{a\}, \{\longrightarrow_{SL} a, a_{out} \longrightarrow_{SL} a\}>, \{(a, \textit{in})\}>$, $d = \longrightarrow_{SL} a$ and $VA(<\{a\}, \{a_{out} \longrightarrow_{SL} a\}>) = \varnothing$

IDN - as before, the only way this can be guaranteed is if there are no uninterpretable networks.

(K^-2) $K_d^- \subseteq K$:

JTMS - this is parallel to the monotonic expansion postulate K^+3 and is also unsatisfied by the JTMS:

$K = <<\{a, b\}, \{\longrightarrow_{SL} a, a_{out} \longrightarrow_{SL} b\}>, \{(a, \textit{in})\ (b, \textit{out})\}>$

$d = \longrightarrow_{SL} a$

$K_d^- = <<\{a, b\}, \{a_{out} \longrightarrow_{SL} b\}>, \{(a, \textit{out})\ (b, \textit{in})\}>$

$K_d^- \not\subseteq K$.

IDN - similarly, K^-2 is satisfied only if all the summation functions concerned are monotonic.

(K^-3) $\alpha \notin K \Rightarrow K_\alpha^- = K$:

JTMS/IDN - satisfied

(K^-4) $\nvdash \alpha \Rightarrow \alpha \notin K_\alpha^-$:

JTMS/IDN - no dependency is tautological (i.e. in every network in every belief state) and the postulate is guaranteed to hold as long as K_d^- is a well defined belief state.

(K^-5) $\alpha \in K \Rightarrow K \subseteq (K_\alpha^-)_\alpha^+$:

JTMS/IDN - this succeeds iff there are no ambiguous networks or the addition of a dependency is governed by a selection function *Sel* that deterministically produces

a single belief state for any given network.

E.g. if $VA(\langle \mathbf{N},\mathbf{D}\rangle_K) = \{V_1, V_2\}$, $VA(\langle \mathbf{N},\mathbf{D}\rangle_{K_d^-}) = \{W_1, W_2\}$ then

$\langle\langle \mathbf{N},\mathbf{D}\rangle_{K_d^-}, W_1\rangle_d^+ = Sel(V_1, V_2)$ must equal $Sel(V_1, V_2) = \langle\langle \mathbf{N},\mathbf{D}\rangle_{K_d^-}, W_2\rangle_d^+$

(K^-6) $\vdash \alpha \leftrightarrow \beta \Rightarrow K_\alpha^- = K_\beta^-$:

JTMS/IDN - there is no notion of logical equivalence for dependencies and so it is not possible to evaluate this criteria.

In addition to the postulates considered above, the principle of positive recoverability (PR), i.e. $(K_d^+)_d^-$, was highlighted in Chapter 1 as a desirable property of belief revision systems. However, as with K^-5, an IDN in general and a TMS in particular will only satisfy PR if the selection function *Sel* determines the result of the contraction deterministically from the set of possible belief states. I.e. for every set $VA(\langle \mathbf{N},\mathbf{D}\rangle)$ there is a preferred admissible valuation. This appears to be quite a pessimistic result but less strict than, for example, defining a total order of preferred belief states.

In general if the summation functions are all monotonic, every network is interpretable and ambiguous networks are consistently interpreted in the same way, then an IDN will satisfy Gardenfors' belief revision postulates at the network level, i.e. when the desired changes or *epistemic inputs* to a belief state are specified in terms of changes to the dependency network.

However, the user may be more interested in getting the network to reflect certain values and in making changes at the valuation level. The obvious way to translate assertions of values for nodes into changes in a dependency network is to add the appropriate assertional dependency:

$\langle\langle \mathbf{N},\mathbf{D}\rangle, V\rangle_{(n,v)}^+ = \langle\langle \text{nodes}, \mathbf{D}', V'\rangle$

where $V' \in VA(\langle \mathbf{N}',\mathbf{D}'\rangle)$, $\mathbf{D}' = \mathbf{D} \cup \{\longrightarrow_v n\}$

In the case of wanting to assert a value for nodes that previously had a *nil* value this is guaranteed success and if the IDN obeys K^+1–6 at the dependency level then K^+1–6 will be obeyed at the valuation level. If a node has a value $V(n) = v_1 \neq nil$ then the addition of an assertional dependency $\longrightarrow_{v_2} n$ may be insufficient to get $V'(n) = v_2$[12]. In this case it may

[12] If the set of values $\mathcal{V}$ is ordered and the node summation function S^N is a maximising function then it is always possible to increase the value of n to v_2 by adding an assertional dependency $\longrightarrow_{v_2} n$.

be necessary to revise the network by retracting existing support and then asserting a new value.

This is a reflection of the fact that support for a proposition is summed over all of the possible sources of support, and the assertion $\longrightarrow_{v_2}$ is not treated preferentially in assigning n a value. If an absolute assertion is needed, i.e. despite all other evidence or support n has value v_2, then this ability must be built-in via the interpretation structure $<\mathcal{V}, S>$. This must be done with caution for if assertions are allowed to override existing evidence without actually revising the network in any other way there is a danger that counter-intuitive results will be produced.

For example, imagine an admissible interpretation of an IDN based on ATMS values where a node has a label {{A B} {C}}. Also imagine the user knows (either through observation or experience) that node should have a value {{D}}, given the possible worlds represented by {A B}, {C} and {D}. This discrepancy between the admissible value and the user's perceived value indicates a mismatch between the structure of the dependency network and the user's expectation. It would be more productive if that mismatch were addressed directly by changing either the assumptions, or the structure of the network, rather than compensating (through absolute assertions) for the mismatch. This means that the root cause of the mismatch is tackled, possibly rectifying other unobserved mismatches, rather than covered up.

Retraction of support is only possible by removing supporting dependencies from the network, be this done under user control or through automatic procedures. If the value to be retracted is specified by the user either directly or is indirectly a part of a revision then a set of dependencies to be retracted will be specified by a backtracking procedure. Having removed these dependencies in a manner transparent to the user so that the required value is retracted, adding an assertional dependency to reassert that value will in most cases result in a different belief. Even if the low-level retraction of dependencies obey K^-1–6 there is no guarantee the high-level retraction of values will obey the postulates. In particular K^-5 will not be satisfied unless retraction is the inverse of assertion.

For example, take the following network and admissible valuation which as a pair constitute a belief state: **<N,D>**

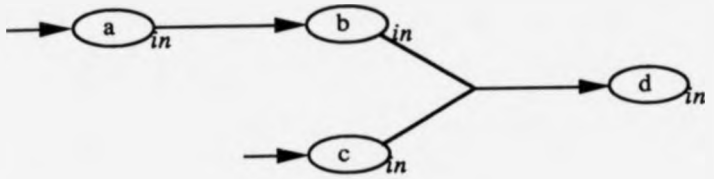


Figure 3.5 (a)

V = {(a, *in*) (b, *in*) (c, *in*) (d, *in*)}

Let **<<N,D>**, $V>^{-}_{(d, in)}$ = **<<N′,D′>**, V′>:

<N′,D′>

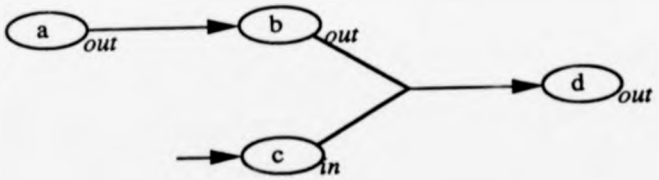


Figure 3.5 (b)

V′ = {(a, *out*) (b, *out*) (c, *in*) (d, *out*)}

Let $(\langle\langle N,D\rangle, V\rangle^{-}_{(d,in)})^{+}_{(d,in)} = \langle\langle N'',D''\rangle, V''\rangle$:

$\langle N'',D''\rangle$

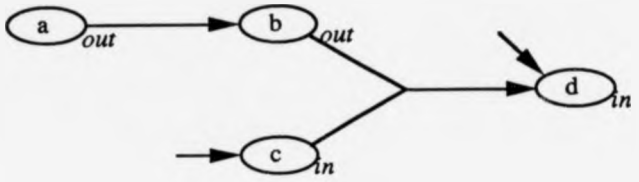


Figure 3.5 (c)

$V'' = \{(a, out)\ (b, out)\ (c, in)\ (d, in)\} \neq V$

Obviously, given $V'' \neq V$, $K^{-}5$ is not satisfied. However, looking at the dual property of positive recoverability, consider a (successful) assertion of a value v for n (i.e. n had a *nil* value prior to the expansion) through the addition of an assertional dependency $\longrightarrow_{v}n$. In order to successfully retract v this dependency must be removed and if the postulates for expansions and contractions are satisfied at the network level then positive recoverability will be satisfied.

In conclusion, in order to satisfy the Gardenfors' postulates at a network level an ordering must be defined on $\mathcal{V}$ and the summation functions must be monotonic. Additionally, each network must be interpretable and have a single preferred admissible valuation if ambiguous. This guarantees satisfaction of all postulates as well as positive recoverability. However, satisfying all postulates at the network level does not mean that additions and retractions at the valuation level will also satisfy the postulates. In particular, unless retractions and expansions by the same value produce identical sets of dependencies to subtract and add to the network then $K^{-}5$ will not be satisfied. This is the result that was argued for in Chapter 1 when weighing the pros and cons of a foundations approach versus a coherence approach to belief revision.

3.6 Monotonicity

In Chapter 1 I introduce the notion of non-monotonic logics and in Chapter 2 I review TMSs as devices for implementing non-monotonic reasoning. In the light of the reconstruction of TMSs as IDNs and the definitions of entailment, I now want to look at the monotonicity, or otherwise of the resulting systems. Before doing this, I want to look at the sources of non-monotonic behaviour.

To recap, $\mathcal{M}(\alpha)$ is defined to be the set of models of α, i.e. those semantic structures satisfying α, $\mathcal{M}(\alpha) = \{ S \in \mathcal{C} \mid S \models \alpha \}$ where $S \models \alpha$ if $i(\alpha, S)$ = true. A system is non-monotonic (described in model theoretic terms) if $\exists\ \alpha, \beta, \gamma$ s.t. $\alpha \models \beta$ but $\alpha, \gamma \not\models \beta$ i.e. $\mathcal{M}(\alpha) \subseteq \mathcal{M}(\beta)$ but $\mathcal{M}(\alpha, \gamma) \not\subseteq \mathcal{M}(\beta)$.

There are two potential sources of non-mononicity. The first is that models satisfying α and γ together ($\mathcal{M}(\alpha, \gamma)$) are not those that satisfy both α and γ individually. I.e. $\mathcal{M}(\alpha, \gamma) \neq \mathcal{M}(\alpha) \cap \mathcal{M}(\beta)$ or more generally $\mathcal{M}(\alpha, \gamma) \not\subseteq \mathcal{M}(\alpha)$. So introducing γ does more than just add a restriction on the models satisfying α, as is the case in monotonic logics. An example of this is Shoham's work on preferences [Shoham 1988] where the models preferentially satisfying α and γ may not be a subset of those models preferentially satisfying just α.

Alternatively, the selection of models satisfying α might not be independent of the interpretation of other propositions, i.e. $\mathcal{M}(\alpha) = f[\mathcal{M}(\gamma_1), ..., \mathcal{M}(\gamma_n)]$. An example of this is in non-monotonic logics that depend on fixed points. In NML-1 [McDermott and Doyle, 1980] a non-monotonic model of A is defined as a pair <V, S> where V is an interpretation of S s.t. $\forall p \in S$, V(p) = 1 [i.e. V is a (monotonic) model of S], and

$$S = \mathrm{Thm}(A \cup AS_A(S)) \text{ where } AS_A(S) = \{ Mq \mid \neg q \notin S \} \setminus \mathrm{Thm}(A).$$

So $\mathcal{M}(\alpha)$ is dependent on all of A, where A is some given theory.

Given standard IDN entailment over a network **<N,D>**, $\mathcal{M}(\alpha) = V \in VA(\mathbf{<N',D'>})$ s.t. $\mathbf{<N',D'>} \in E(\mathbf{<N,D>})$ and $i(\alpha)$ = true. As $\mathcal{M}(\alpha, \gamma) \subseteq \mathcal{M}(\alpha)$, and $\mathcal{M}(\alpha)$ is independent of the interpretation of β, depending only on **<N',D'>**, $\models_{IDN}$ is a monotonic operation. This is highlighted by considering some theorems and non-theorems. If $n_{in}, m_{out} \longrightarrow_{SL} l$ were to be viewed as a non-monotonic inference of l from $\neg$m conditioned on n then one would expect

the following theorems

nThm $n_{in}, m_{out} \longrightarrow_{SL} l$, (n, *in*) $\models_{IDN}$ (l, *in*) and

nThm $n_{in}, m_{out} \longrightarrow_{SL} l$, (n, *in*), (m, *in*) $\not\models_{IDN}$ (l, *in*)

However, this is not the case - consider an interpretation of the antecedents of the theorem where the dependency $\longrightarrow_{SL} m$ is added to the network - the antecedents for the first (non-)theorem are satisfied, yet the consequent is false.

In order to get a valid relationship between n and l, given $n_{in}, m_{out} \longrightarrow_{SL} l$ it is necessary to strengthen the antecedents of the theorem:

nThm $n_{in}, m_{out} \longrightarrow_{SL} l$, (n, *in*), (m, out) $\models_{IDN}$ (l, *in*).

This demonstrates the point that the non-monotonicity in an interpretation must come from some assumption about the default value of nodes as *out*. It is not possible to have the non-monotonicity as a product of the entailment relationship.

3.7 Conclusion

Of the work contained in this thesis, the first two sections (§3.1, §3.2, and §3.3) expound and detail the core idea contained in the thesis: that each node should be assigned a level of belief that corresponds to the explicit support for that node. §3.1 articulates the semantics of support in an informal way and motivates the need for a formal semantics; §3.2 provides the basic definition of an IDN in terms of a network structure with an interpretation mechanism.

Section 3.3 is more speculative and examines notions of entailment in an IDN in an attempt to show how the individual dependencies can be thought to represent entailments in a model theoretic sense. This section also explores some of the implications of, or reasons behind, certain approaches taken in the design of IDNs.

The remainder of the chapter looks backwards. §3.4 shows how the TMSs presented in Chapter 3 can be reinterpreted as IDNs, with certain provisos on the functionality replicated and added conditions on the admissible valuations so that they conform to the semantics (informal or otherwise) of the existing systems.

Section 3.5 shows how Gardenfors' postulates for belief revision (GBPR) can be applied at two different levels when talking of IDNs. At the network level there are strict conditions that must be met for the addition and retraction of dependencies to satisfy GPBR. Even if this is guaranteed this is insufficient for the IDN to satisfy GBPR at the interpretation level. At this level the changes of interest are in the valuation of a particular network and are brought about by changes to the network. Unless retractions and assertions map uniformly to changes in the dependency structure then GBPR will not be satisfied.

Finally §3.6 examines one of the surprising consequences of IDNs: that non-monotonic inference, expressed in model theoretic terms, is not possible within the IDN framework. It is not possible to prove a relationship between two or more propositions, and have that relationship disproved by the addition of more evidence. However it is possible to prove relationships (i.e. one of support) between propositions, given that relationship includes explicit reference to missing or unknown information.

CHAPTER 4

Construction of Valuations

The aim of Chapter 3 has been to lay the foundations for what follows in Chapter 4. In Chapter 3 I defined an Interpreted Dependency Network as a network of dependencies with a set of functions prescribing what valuations should be constructed. This interpretation mechanism forms the basis of a declarative semantics, based on admissible valuations, which is used to define several notions of entailment. This in turns supports the idea of networks as background theories with individual dependencies capturing particular inferences or entailments. Finally I show that existing TMSs can be reconstructed as equivalent (or near equivalent) IDNs.

All the discussion of semantics takes for granted the existence of the set of admissible valuations of a network, $VA(\langle \mathbf{N},\mathbf{D} \rangle)$, without reference to how the valuations are constructed. This work will form the main body of this chapter. In particular in §4.1 I look at the construction of valuations for acyclic networks and in §4.2 for cyclic networks. In §4.3 I look at how different modes of reasoning can be interpreted as different ways of manipulating a network and its set of admissible valuations.

Before actually looking at the construction of admissible valuations I want to further examine why they are important. In particular I want to relate their properties to those of belief sets as outlined by Gardenfors [1988].

4.0.1 Admissible Valuations

As outlined in §1.2, Gardenfors defines a belief set to be a set of propositions that is closed under some inference scheme and consistent. Expansion, contraction and revision functions are defined as operations on belief sets, mapping one belief set to another given some epistemic input.

Admissible valuations are analogous to belief sets in the context of IDNs, being both closed and consistent. Viewing dependencies as inferences, when the value of the consequent does not correspond to that supported by the values of the antecedents, then we can view the dependency as a valid inference whose conclusion has yet to be asserted. Such valuations are not admissible because $V(n) \neq V^{\mathbf{N}}(n, V)$. For example, given two valuations V_1 and V_2 representing two belief sets before and after an inference is made:

knowledge before (V_1)	**inference**	**knowledge after** (V_2)
a, a $\rightarrow$ b	a, a $\rightarrow$ b $\models$ b	a, a $\rightarrow$ b, b
(a, *in*) , (b, *out*)	(a, *in*) , a $\longrightarrow_{SL}$ b $\models_{<N,D>}$ (b, *in*)	(a, *in*) , (b, *in*)

V_1 (b) = *out* $\neq$ *in* = V^N(n, V_1) - V_1 is therefore not an admissible valuation. This demonstrates how admissible valuations can be viewed as a closed set of inferences.

Similarly, admissible valuations can be viewed as consistent. They are internally consistent in that they do not violate the summation functions and can be viewed as obeying a set of truth functional composition rules. Additionally, given some external criteria for consistency, admissible valuations should not be labelled as inconsistent, be it by having a JTMS contradiction node valued as *in* or by having an ATMS no-good contained in an environment of some node's label.

4.0.2 Theory Questions

Given the identification of admissible valuations both as interpretations of a set of information (represented by assertional dependencies) supported by a particular theory and as rational belief states, finding the admissible valuations is, as stated before, one of the main processes to be investigated in this thesis. There are other questions I wish to address that correspond to other modes of reasoning. Below I shall frame these questions both in relation to IDNs and give an intuitive reading of the problem. Where appropriate I will also refer to the belief revision criteria or operations as outlined by Gardenfors.

Closure

Q1 Find the admissible valuations of a network, $VA(\langle \mathbf{N},\mathbf{D} \rangle)$

≡ Given a set of propositions Γ find its transitive closure Σ, i.e. $\Sigma \supseteq \Gamma$ s.t. $Cn(\Sigma) = \Sigma$

≡ Given a background theory ($\langle \mathbf{N},\mathbf{D} \rangle$ or Γ) find its consistent closed interpretations.

Expansions

Q2 Given a set of dependencies $D \subseteq \mathcal{D}$, and a set of admissible valuations $VA(\langle \mathbf{N},\mathbf{D} \rangle)$, find $VA(\langle \mathbf{N},\mathbf{D} \cup D \rangle)$

≡ Given a set of consistent closed interpretations of a given theory find the set of similar interpretations of an extended theory and/or set of observations.

Contractions

Q3 Given a set of dependencies $D \subseteq \mathcal{D}$, and a set of admissible valuations $VA(<\mathbf{N},\mathbf{D}>)$, find $VA(<\mathbf{N},\mathbf{D} \setminus D>)$

≡ Given a set of consistent closed interpretations of a given theory find the set of similar interpretations of a contracted theory and/or set of observations.

Explanations

Q4 Given V_p, find $V_a \in VA(<\mathbf{N},\mathbf{D}>)$ s.t. $|V_a \cap V_p|$ is maximal

≡ find $\Sigma \supseteq \Gamma$ s.t. $Cn(\Sigma) = \Sigma$ and $|\Sigma \cap \Phi|$ is maximal

≡ Given a set of facts (V_p or Φ), find the interpretation of the theory that is closest to or accords best with that set of facts.

Q5 Given V_p find the minimal $D \subseteq \mathcal{D}_a$ s.t. $V_p \in VA(<\mathbf{N},\mathbf{D} \cup D>)$

≡ Given a set of facts (V_p or Φ) find the smallest set of additional assumptions (D or Δ) that has to be made to support that set of facts.

Of these questions: the first is addressed in §5.1 and §5.2 - the interpretation of acyclic and cyclic networks; the second, third and fourth are addressed in §4.3.1 - the forward propagation of changes; and the last two are covered in §4.3.2 - query or goal-driven processing.

In general, we may not want or need *all* of the possible admissible valuations but may want to work within and maintain just one[1]. By restricting ourselves to single elements of $VA(<\mathbf{N},\mathbf{D}>)$, expansion (+) and contraction (−) operations can be defined so that they take a network, one of its admissible valuations and a set of dependencies to be added or deleted and return a new network and a single admissible valuation. As network/valuation pairs form the class of semantics structures we can think of these operations as being defined on $\mathcal{C} \times Pw(\mathcal{D}) \longrightarrow \mathcal{C}$ with

[1] Even if that valuation has a possible worlds semantics and therefore represents more than one possible interpretation at the semantic level.

$(\langle \mathbf{N},\mathbf{D}\rangle, V)^{+}_{D} = (\langle \mathbf{N},\mathbf{D} \cup D\rangle, V')$ where $V' \in VA(\langle \mathbf{N},\mathbf{D} \cup D\rangle)$

$(\langle \mathbf{N},\mathbf{D}\rangle, V)^{-}_{D} = (\langle \mathbf{N},\mathbf{D} \setminus D\rangle, V')$ where $V' \in VA(\langle \mathbf{N},\mathbf{D} \setminus D\rangle)$

If $\langle \mathbf{N},\mathbf{D}\rangle$ is implied by the context it can be omitted from the equation so that $(\langle \mathbf{N},\mathbf{D}\rangle, V)^{+}_{D}$ becomes V^{+}_{D}. The following properties (introduced in §1.2.3.1) can then be defined:

Defn A valuation V is *positively recoverable* w.r.t a set of dependencies D (and a given network $\langle \mathbf{N},\mathbf{D}\rangle$) iff $(V^{+}_{D})^{-}_{D} = V$.

Defn A valuation V is *negatively recoverable* w.r.t a set of dependencies D (and a given network $\langle \mathbf{N},\mathbf{D}\rangle$) iff $(V^{-}_{D})^{+}_{D} = V$.

and the following questions are interesting:

Q7 Under what conditions does $((\Sigma^{+}_{\alpha})^{-}_{\alpha}) = \Sigma$

≡ Given the interpretation of a theory, which is then extended by some premises, when does the removal of those premises return the interpretation to its original state?

Q8 Under what conditions does $((\Sigma^{-}_{\alpha})^{+}_{\alpha}) = \Sigma$

≡ Given the interpretation of a theory, which has some information retracted, does the subsequent reintroduction of that information return the interpretation to its original state?

Gardenfors claims that this relationship should be valid for non-probabilistic belief revision, though I claim (see §1.2.3.1) that for all relevant or causal systems this cannot and **should not** hold.

The question about positive vs negative recoverability has already been dealt with in §3.5 and is only included here for completeness. If α is a dependency, then positive and negative recoverability will only hold if all networks have a single admissible valuation, or failing that a selection function that always returns the same admissible valuation. If α is a desired change in a node's value, e.g. $\alpha \equiv$ "make n *in*", then positive and negative recoverability will only hold if the changes in the network needed to assert the appropriate value are the same as those needed to retract that value. Although this is easily managed for positive recoverability, the same is not true for negative recoverability.

4.1 Interpretation of Acyclic Networks

The distinction between networks that allow cycles and those that do not is an obvious one to make. I make this distinction because the interpretation of the former is easier while the latter are much harder to interpret. This distinction can be made in a wide variety of knowledge representations, including some not explicitly defined in terms of networks. Having made this distinction, many of these representations expressly forbid cyclic networks, for much the same reasons that this distinction is made in IDNs, namely efficiency. Examples of such representations include:

- Stratified Databases [Apt and Pugin, 1987], a form of deductive database intended for non-monotonic reasoning that does not allow for recursion through negation;
- Knowledge Based Systems, e.g. through the avoidance of circular rules sets in ART [Nguyen 1987];
- Bayesian Networks [Pearl 1988] used for representing statistical causal models;
- Reason Maintenance Systems [Falkenhainer 1987] based on Dempster-Schafer belief revision; and
- Inheritance systems, e.g. frame-based systems.

In particular, Touretzky [1986] states

> "All inheritance systems with which I am familiar require the inheritance graph to be IS-A acyclic"

citing reasons of computational tractability and semantic clarity, i.e. cycles lack a clear semantics. He claims generality for his system because he does not require networks to be acyclic, yet most of his "major theorems" (his term) depend on this, e.g. Thms 2.7, 2.11, 2.13, 2.14, and corollary 2.9 [p209-210].

In the rest of this section I shall give an algorithm for calculating admissible valuations of acyclic dependency networks, and prove that it terminates having produced all possible correct answers, i.e. that the algorithm is sound and complete. Before actually doing this, I need to formally define "acyclic dependency networks".

Defn $pars(n) = \bigcup_{d \in D(n)} ants(d)$ - the parent nodes of n

$ancs(n) = \{ x \mid \exists\ p, \text{ s.t. } x \in pars^{p}(n) \}$ - the ancestors of n

Defn A network **<N,D>** is cyclic iff $\exists\ n \in \mathbf{N}$ s.t. $n \in ancs(n)$

otherwise a network is acyclic.

Defn For $n,m \in \mathbf{N}$ n *precedes* m , $n < m$, iff $n \in ancs(m)$

Alternatively n is said to be *upstream* of m, and m is *downstream* of n.

Given two sets of nodes N and M, $N < M$ iff $\forall n \in M, m \in M, n < m$

Lem A network is cyclic iff $\exists$ n,m s.t. $n < m$ and $m < n$ For an acyclic network, $\forall$ n,m exactly one of the following holds:

(i) $n < m$

(ii) $m < n$

(iii) neither $n < m$ nor $m < n$

4.1.1 Algorithm

The algorithm in this section basically works by doing a breadth-first calculation of values for nodes. The algorithm works by keeping for each node $n \in \mathbf{N}$ a record in the array NP of the number of parents of n that have not had their values calculated. When all the parent's values have been calculated, i.e. NP[n] = 0, then n has its value calculated using the appropriate summation functions applied to the values of the antecedents of n dependencies. These values are found in V which is used to store nodes' values as they are calculated and is the resulting admissible valuation produced by the algorithm[2]. The consequents of n then have their NP reduced by 1. The nodes whose values are to be calculated in each iteration are stored in TC and during that iteration, the TC for the next iteration are stored in TCN.

The algorithm takes a network **<N,D>** and a partial valuation V0. The partial valuation is used when calculating admissible valuations for acyclic components of a cyclic network

[2] Notice that V is only a partial valuation when calculating all but the last node but this has no effect on the calculation as all the necessary information is present. In any case, V can be converted into a total valuation simply by adding a *nil* value for all nodes whose values have yet to be calculated.

and represents some previously derived values for root nodes of an acyclic component. For totally acyclic networks this is irrelevant.

Find-Acyclic-VAs(<**N**,**D**>, V0)

```
TC = ∅                                          {initialise}
for n ∈ N do
    begin
    NP[n] = |pars(n)|                           {no. of parents to be evaluated}
    If NP[n] = 0 then TC = TC ∪ {n}             {nodes to be calculated next}
    end
for (n, v) ∈ V0 do                              {set up initial valuation V0}
    begin
    V[n] = v
    NP[n] = -1
    for m ∈ cons(n) do                          {for each consequent ...}
        begin
        NP[m] = NP[m] - 1                       {reduce no. of parents to be evaluated}
        If NP[m] = 0 then TC = TC ∪ {m}         {set nodes to recalculate}
        end
    end
while TC ≠ ∅ do                                 {main loop}
    begin
    TCN = ∅
    for n ∈ TC do                               {for each node to be calculated ...}
        begin
        V = V ∪ { (n, V^N(n ,V)) }              {calculate value}
        for m ∈ cons(n) do                      {for each consequent ...}
            begin
            NP[m] = NP[m] - 1                   {reduce no. of parents to be evaluated}
            If NP[m] = 0 then TCN = TCN ∪ {m}   {set nodes to recalculate}
            end
        end
    TC = TCN
    end
return {V}
```

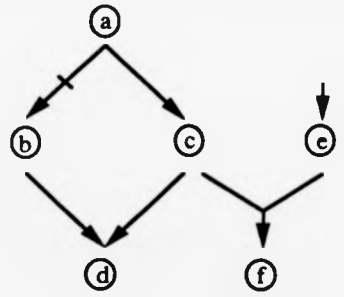
In calculating the complexity I have assumed that *pars* and *cons* have already been calculated, either when the network was first constructed, or as and when new dependencies have been added, so that finding *pars*(n) or *cons*(n) are simple table look-ups. Each union operation is guaranteed to involve unique elements, i.e. there is no chance of repetition, so we can assume this is also a linear time operation.

The initialisation process is of order N = |N| The number of iterations of the main loop is equal to the depth, i.e. the length of the longest path, of the network. This is irrelevant as we know the total number of operations without actually knowing **when** they are performed. To calculate V is $N \times [\, O(V^N(n, V)) + 1]$. If the total number of parents of all nodes is

$$P = \sum_{n \in N} |pars(n)|$$

then there are N + P subtractions from NP, and P tests for inclusion in TCN, and N resulting union operations. Finally there are 2D assignments each of the form TCN = ∅ or TC = TCN. The dominant factor in all this complexity is the actual calculations of the values in V, giving an overall complexity of O(N×M) where $M = O(V^N(n, V))$.

Example



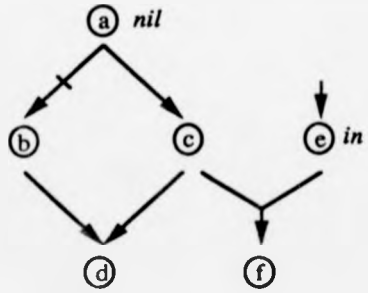
end of initialisation

V = { }

NP = [0, 1, 1, 2, 0, 2]

TC = {a, e}

Figure 4.1.1 (a)



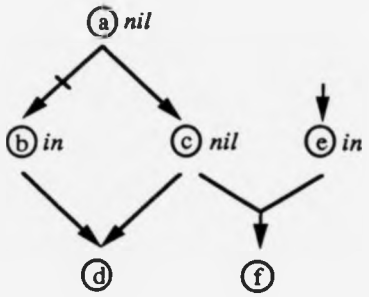
end of first iteration

V = {(a, *nil*) , (e, *in*)}

NP = [0, 0, 0, 2, 0, 1]

TC = {b, c}

Figure 4.1.1 (b)



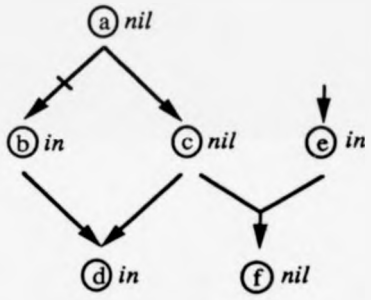
end of second iteration

V = {(a, *nil*) , (e, *in*) , (b, *in*) , (c, *nil*)}

NP = [0, 0, 0, 0, 0, 0]

TC = {d, f}

Figure 4.1.1 (c)



end of third iteration

V = {(a, *nil*) , (e, *in*) , (b, *in*) , (c, *nil*) , (d, *in*) , (f , *nil*) }

NP = [0, 0, 0, 0, 0, 0]

TC = { }

Figure 4.1.1 (d)

4.1.2 Termination

Having given the algorithm, I want to establish three things: that the algorithm terminates and terminates at the right time; that the answer produced is correct, i.e. is an admissible valuation; and that the algorithm is complete, i.e. produces *all* the right answers. In the following, if X is a variable name then X_i stands for the value of X at the start of iteration i.

To prove termination I need the following:

Defn A *path p from n to m of length l* is a set of dependencies $p = \{d_1, ..., d_l\}$ s.t. $n \in ants(d_1)$, $m = cons(d_l)$, $\forall i$, $cons(d_i) \in ants(d_{i+1})$ and $\forall i$, $cons(d_i) \notin ants(d_j)$ for $j < i$.

p is said to *originate* at n and *terminate* at m.

Defn A node n is of *depth d* iff $|p| = d$, where p is the longest path terminating at n

Lem If node n is of depth k, then (i) $\forall m \in pars(n)$, m is of depth $k' < k$ and
(ii) $\exists p \in pars(n)$ s.t. p is of depth k-1.

Proof (i) If $\exists m \in pars(n)$ s.t. m is of depth $\geq k$, then there is a path p of length $k' > k$ terminating at m. As $m \in pars(n)$ $\exists k \in D(n)$ s.t. $m \in ants(d)$ and $n = cons(d)$. So $p' = p \cup \{d\}$ is of path terminating at n, of length $k'+1 > k$, contradicting n being of depth k, so m must be of depth $k' \leq k$.

(ii) If n is of depth d, let p be a path of length k terminating at n. If d_k is the last dependency in p with $m = cons(d_{k-1})$, then $m \in pars(n)$, the path $p \{d^k\}$ terminating at m is of length k-1, and m is of depth k-1.

□

I now prove the following result

Thm For a node n of depth k, V(n) is calculated at iteration k+1.

Proof By induction.

If k (the depth of n) = 0 then $NP_0[n] = 0$ and V(n) will be calculated at the first iteration.

If k = i and we assume that all nodes of depth $k' < k$ are calculated at iteration $k'+1$ then $\forall m \in pars(n)$, given the depth of $m < k$ (by corollary), V(m) has been calculated by the end of iteration $k'+1 < i+1$.

So $NP_{i+1}[n] = 0$ and V(n) is calculated no later than iteration i+1.

By corollary(ii) $\exists m \in pars(n)$ s.t. depth of m is i-1.

Given the inductive hypothesis, V(m) is only calculated at iteration i so $NP_i[n] > 0$ and V(n) must be calculated at iteration i+1.

□

Given **<N,D>** is finite then every node has a finite depth and the algorithm is guaranteed to terminate. Furthermore, as every node has a parent of depth one less than its own, $TC_i \neq \varnothing$ for all i less than the maximum depth and the program only terminates when all nodes have had their values calculated.

4.1.3 Soundness

Having proved that the algorithm terminates at the correct time we need to establish that the valuation produced is in fact admissible. From the above theorem (in §4.1.2) we know that the algorithm iteratively expands the valuation V until it is a complete valuation of **<N,D>**, i.e. $\forall$ n $\in$ **N** V(n) is defined. We also know each node's value is calculated only once - as n $\in$ TCN only once, as soon NP[n] becomes 0. This gives a series of partial valuations $V_1 \subset V_2 \subset$,..., $\subset V_d$ where $V(n) = V_j(n) = V^{\mathbf{N}}(n, V_{i-1})$ if n is of depth j. We can use this to prove the following theorem.

Thm The valuation V produced by the algorithm in §4.1.1 is an admissible valuation.

Proof If the algorithm is unsound then $\exists$ n s.t. $V(n) \neq V^{\mathbf{N}}(n, V)$.

Assume V(n) is calculated at iteration i so then $V(n) = V_i(n) = V^{\mathbf{N}}(n, V_{i-1})$.

If $V(n) \neq V^{\mathbf{N}}(n, V)$ then $V|_{pars(n)} \neq V_{i-1}|_{pars(n)}$.

As values do not change once calculated, this can only happen if V(n) has not been calculated for some parent of n. But we know that all the parents of n are calculated **before** n.

So $V_{i-1}|_{pars(n)} = V|_{pars(n)}$ and therefore $V^{\mathbf{N}}(n, V) = V^{\mathbf{N}}(n, V_{i-1}) = V_i(n) = V(n)$ contradicting the assumption that the algorithm is unsound.

□

4.1.4 Completeness

Having shown that the algorithm terminates with a correct solution, i.e. it produces an admissible valuation, we want to show that it produces all possible correct solutions. This requires the following definitions and lemma.

Defn Given a function mapping one set of node names to another, f:**N**$\longrightarrow$**N′**, the image of a dependency $d = A_1$,..., $A_q \longrightarrow_t c$ under f is $f(d) = A'_1$,..., $A'_q \longrightarrow_t c$ where $\forall n \in \mathbf{N}, n \in A_i \leftrightarrow f(n) \in A'_i$.

Defn Given an isomorphism[3] f:$\mathbf{N} \longrightarrow \mathbf{N'}$, *a dependency* d *is isomorphic to* d′ *under f*, written $d \approx_f d'$, iff $f(d) = d'$. Two sets of dependencies are isomorphic under f, $D \approx_f D'$ iff $\forall d \in \mathbf{D}, d \in D \leftrightarrow f(d) \in D'$.

Lemma (Isomorphic Parents)

Given networks <**N,D**> and <**N′,D′**> and valuations V_1, V_2 respectively,
if $D(n) \approx_f D(m)$ for $n \in \mathbf{N}$, $m \in \mathbf{N'}$ and $\forall p \in pars(n)$, $V_1(p) = V_2(f(p))$
then $V^{\mathbf{N}}(n, V_1) = V^{\mathbf{N}}(m, V_2)$.

Proof As $D(n) \approx_f D(m)$ then $D(m) = \{f(d_1)\ ,...,\ f(d_q)\}$ for $D(n) = \{d_1\ ,...,\ d_q\}$ where $f(d_i) = A'_1\ ,...,\ A'_r$ for $d_i = A_1\ ,...,\ A_r$ and $A'_j = \{f(a_1)\ ,...,\ f(a_s)\}$ for $A_j = \{a_1\ ,...,\ a_s\}$.

As $\forall p \in pars(n)$, $V_1(p) = V_2(f(p))$ then $\forall a_k \in A_j$, $V_1(a_k) = V_2(f(a_k))$.

So $\forall A_j \in d_i$, $V^{\mathbf{A}}(A_j, V_1)) = S^{\mathbf{A}}(V_1(a_1)\ ,...,\ V_1(a_s))$

$$= S^{\mathbf{A}}_{f(\mathbf{A})}(V_2(f(a_1))\ ,...,\ V_2(f(a_q))) = V^{\mathbf{A}}(A'_j, V_2))$$

and $\forall d_i \in D(n)$, $V^{\mathbf{D}}(d_i, V_1) = S^{\mathbf{D}}(V^{\mathbf{A}}(A_1, V_1)\ ,...,\ V^{\mathbf{A}}(A_p, V_1))$

$$= S^{\mathbf{D}}(V^{\mathbf{A}}(A'_1, V_2)\ ,...,\ V^{\mathbf{A}}(A'_p, V_2)) = V^{\mathbf{D}}(f(d_i), V_2)$$

and therefore $V^{\mathbf{N}}(n, V_1) = S^{\mathbf{N}}(V^{\mathbf{D}}(d_1, V_1)\ ,...,\ V^{\mathbf{D}}(d_m, V_1))$

$$= S^{\mathbf{N}}(V^{\mathbf{D}}(f(d_1), V_2)\ ,...,\ V^{\mathbf{D}}(f(d)_m, V_2)) = V^{\mathbf{N}}(m, V_2).$$

□

Using this lemma the following theorem can be proved.

Thm If <**N,D**> is acyclic then $|VA(\langle \mathbf{N,D} \rangle)| = 1$.

Proof Assume that $|VA(\langle \mathbf{N,D} \rangle)| > 1$. Without loss of generality,

let $VA(\langle \mathbf{N,D} \rangle) = \{ V_1, V_2 \}$ where $V_1 \neq V_2$.

Let $N_d = \{ n \in \mathbf{N} \mid V_1(n) \neq V_2(n) \}$ and assume that $N_d \neq \varnothing$.

Given <**N,D**> is acyclic, $\exists n \in N_d$ s.t. $pars(n) \cap N_d = \varnothing$ **

i.e. $\forall m \in pars(n)$, $V_1(m) = V_2(m)$.

As V_1 and V_2 are admissible, $V_1(n) = V^{\mathbf{N}}(n, V_1)$ and $V_2(n) = V^{\mathbf{N}}(n, V_2)$.

Thus, by the Isomorphic Parent Lemma using the identity function as the

[3] [bijection or 1-1 onto map]

isomorphism, $V^{\mathbf{N}}(n, V_1) = V^{\mathbf{N}}(n, V_2)$ and therefore $V_1(n) = V_2(n)$ and $n \notin N_d$ contradicting **.

Therefore N_d is empty, $V_1 = V_2$ and $|VA(\langle \mathbf{N},\mathbf{D} \rangle)| = 1$.

□

As our algorithm produces a correct answer, and it is the only answer, then the algorithm is complete. We also get the following, obvious and simple, corollaries which I shall discuss in further detail in §X.

Cor If <**N,D**> is unsatisfiable, i.e. $|VA(\langle \mathbf{N},\mathbf{D} \rangle)| = 0$, then <**N,D**> is cyclic.

Cor If <**N,D**> has multiple admissible valuations, i.e. $|VA(\langle \mathbf{N},\mathbf{D} \rangle)| > 1$ and is therefore ambiguous then <**N,D**> is cyclic.

4.1.5 Discussion

The above theorem, in §4.1.4, states that an acyclic network has just a single admissible valuation. This is an important result, above and beyond the fact that it proves the algorithm for interpreting acyclic networks is complete.

Given a particular network <**N,D**>, the extensions of that network

$$E(\langle \mathbf{N},\mathbf{D} \rangle) = \{\langle \mathbf{N},\mathbf{D} \cup D \rangle \mid D \subseteq \mathbf{D}_a, \text{ s.t. } \forall n \in \mathbf{N}, |\ D \cap \mathbf{D}(n)| \leq 1\}$$

each have a unique interpretation so that there is an isomorphism between the assertions D of a given extension and the valuation that results. Thus the set of assertions that are added to a given background theory represented by <**N,D**> completely determines the interpretation of that theory.

Additionally, the uniqueness of admissible valuations means any interpretation is both positively and negatively recoverable at the network level. If we mistakenly assert a proposition using $d \in \mathbf{D}_a$ given some admissible valuation V, then the removal of d from $\langle \mathbf{N},\mathbf{D} \cup D \rangle$ given V' will result in V again. Unfortunately, this useful result does not hold for cyclic networks as we shall see in the next section.

Finally, before going on to calculate admissible valuations for cyclic networks it is important to look at the complexity of calculating acyclic valuations. In §4.1.1 it was shown

to be of order $O(N \times M)$ where $N = |\mathcal{N}|$ and M is the complexity of calculating a single node's value. For something like the JTMS-IDN which involves a set number of single operations over the antecedent set, M is merely linear in the number of antecedents and the overall complexity of system is less than $O(N^2)$ which is (as the consensus goes) significantly different than the NP-complexity proved for general TMSs networks [Elkan 1989] - i.e. for networks that are not restricted to having no cycles. For the ATMS-IDN this rises substantially. If an average dependency has n antecedents, each with l_i environments in its label, then constructing the set of combined environments for the antecedents (i.e. the sets of assumptions that would satisfy all the antecedents) requires (in the worst case):

- $l' = l_1 \times ... \times l_n \times$ set unions, each involving n environments;
- $l'!$ subset tests to produce l'' minimal environments; and $l'' \times m$ subset test to produce a set of consistent environments where m is the number of no-good (inconsistent) environments.

This complexity result, at least in this area, vindicates the supposition made in §4.1 that cyclicity is a computationally expensive representation property. In the next section I hope to show just how expensive it actually is.

4.2 Interpretation of Cyclic Networks

Given the interpretation of acyclic networks is relatively easy, while the interpretation of cyclic networks is difficult (NP-complete), why would one want to allow cyclic networks? As mentioned in §4.1 many knowledge representation schemes freely do make this restriction because of the difficulties either of saying what cycles actually represent, or difficulties in actually reasoning with them.

In the next section I put forward some arguments for allowing cycles and give some intuitive ideas about the meaning of such cycles. In §4.2.2 an algorithm is presented that decomposes a cyclic network into a set of acyclic networks before interpreting them using the algorithm given in §4.1.1. Section 4.2.3 then goes on to look at the termination and complexity. Finally in §4.2.4 I look at the soundness and completeness of the algorithm.

4.2.1 Why Have Cycles?

There are at least three reasons for allowing cycles in patterns of inferences as represented by dependency networks, though they all reduce to a single argument. That is that cycles are naturally occurring phenomena and without them a representation technique lacks a certain expressive power.

There could be a direct correspondence between the value of two nodes. This could represent either an equality or equivalence as in V(a) = *in* iff V(b) = *in* and V(a) = *out* iff V(b) = *out*. Alternatively it could be a inverse relationship V(a) = *in* iff V(b) = *out* and V(a) = *out* iff V(b) = *in*.

Figure 4.2.1 (a)

Two separate lines of arguments could overlap to produce a cycle. For example, if Tweety is a bird and Tweety is a normal bird with respect to flying, then Tweety flies. Conversely and independent of the preceding statements, if Tweety flies and Tweety is a bird then Tweety is a normal bird with respect to flying.

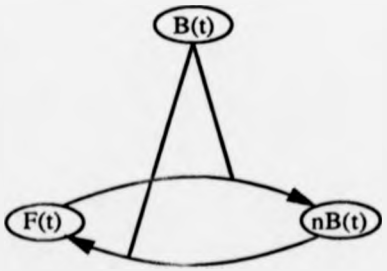


Figure 4.2.1 (b)

Finally there are situations where feedback occurs. A classic example being the case where an individual believes they are no good at a particular activity, this lack of belief in their ability leads to failure, and that failure reinforces the initial belief in their lack of ability. Structurally this is the same as the first example, but there is an added temporal dimension.

4.2.2 Algorithm

The algorithm for calculating admissible valuations for networks that include cycles has three distinct parts. The first task is to decompose the network into a partially ordered set of subnetworks (components) so that each subnetwork is either acyclic or strongly connected (i.e. for any two nodes $n,m \in \mathbf{N}$ there is a path from n to m in $\langle \mathbf{N},\mathbf{D} \rangle$). Strongly connected components are then converted to acyclic networks by generating a *cutset* and duplicating the nodes in it so that no cycles remain. Additionally, as values in a cycle can be self-supporting, the cut-points can be assigned arbitrary values and as long as they are equal to the resulting values of their doubles, those values will be supported and the valuation generated will be admissible. The arbitrary values for cut-points are produced by adding a set of assertional justifications, produced by the *permute* function, to the de-cycled component prior to interpretation.

Each component in turn is then interpreted using the algorithm presented in §4.1.1 with a check performed on each cyclic component to ensure that admissible valuations of the corresponding acyclic network are admissible valuations of the cyclic component. As each component is interpreted the resulting valuations are merged with the valuations from preceding components until a set of admissible valuations for the whole network has been constructed.

The basic algorithm for constructing admissible valuations is given below with the decomposition process explained in §4.2.2.1 and the interpretation of cycles explained in §4.2.2.2 and §4.2.2.3.

Find-VAs(<**N,D**>)

```
VAs ← {∅}                                                          initialise
Components, type ← Decompose-Network(<N,D>)
for <N,D⋃D> ∈ Components do
   begin
   NVAs ← ∅
   if type(<N′,D′>) = "acyclic"
      then for V ∈ VAs do
         NVAs ← NVAs ∪ Find-Acyclic-VAs(N, ∅, V)
      else do
         begin                                                     Start of
         <N′,D′>, C ← Decycle(<N′,D′>)                             Find-Cyclic-VAs
         for D ∈ Permute(<N′,D′>) do
            for V ∈ VAs do
               begin
               NVs ← Find-Acyclic-VAs(<N,D⋃D>, C, V)
               for NV ∈ NVs do
                  If Admissible(NV, <N′,D′>, C, V)
                     then NVAs ← NVAs ∪ NV
               end
         end                                                       End of
   if NVAs = ∅ then return nil else VAs ← NVAs                     Find-Cyclic-VAs
   end
return VAs
```

The code between "Start" and "End" of *Find-Cyclic-VAs* could have been defined as a separate procedure but has been kept in the main body of *Find-VAs* to improve readability and continuity. However, it is sometimes useful in what follows to use the term *Find-Cyclic-VAs* to refer to the procedure of de-cycling a cyclic component, permuting sets of assertional dependencies, interpreting the resulting acyclic network and checking the admissibility of the valuation.

4.2.2.1 Decomposition of Networks

The purpose of the function *Decompose* is to take a network and return an order set of components corresponding to cyclic and acyclic components, along with a type specification for each component. The function works by first identifying all the strongly connected components and then ordering the components according to depth, aggregating single node components to form acyclic networks.

The generation of strongly connected components is a standard graph theory problem and can be solved using an already known algorithm, e.g. *DFSSCC* [Gibbons 1985, p 30]. This algorithm has complexity O(max(|**N**|, |**D**|)) for a network **<N,D>** and returns a list of component identifiers (this is an extension to *DFSSCC* but has no impact on complexity as it is merely linear processing of the output as it is generated). A function (indexed list structure) is set up to record which nodes are elements of which components: N(ci) = { n | **CI**(n) = ci) where ci = a component id; where CI is an inverse function defined to label each node with the appropriate component id. Additionally, a component type label (T) is set to "a" (for cyclic) or "c" (cyclic) depending on whether a component has a single node with no loops to itself ("a") or not ("c").

Acyclic components are used as starting points for backward and forward depth-first searches (performed by the functions *search-forward* and *search-backward*) to identify all the nodes that can be aggregated to form a single acyclic component. Nodes in these components are then relabelled with the new acyclic component id taken from the starting acyclic components. As these searches are linear in nodes or edges and each edge and node is only used in a single search, this process will also be linear in the number of nodes.

Each component then has its antecedent (A(ci)) and consequent components (C(ci)) identified, along with the number of antecedent components that need to be ordered (TO) before a component itself can be ordered. This information is then used to identify the starting points for a breadth-first search and to control when a component's order should be calculated. The breadth-first search is the same as the ordering algorithm for nodes in acyclic valuation calculation and is therefore linear. This results in an overall linear time complexity.

```
Decompose (<N,D>):
Component-ids IC, T, N ← DFSSCC(<N,D>)
for ci ∈ Component-ids where T(ci) = "a" and N(ci) ≠ ∅ do
    begin
    search-forward(ci, ci)
    search-backward(ci, ci)
    end
Q ←∅
for ci ∈ Component-ids do
    begin
    TO(ci) ← 0
    for n ∈ N(ci) do
        begin
        for p ∈ pars(n) do
            if CI(p) ∉ A(ci) then do
                begin
                push CI(p) onto A(ci)
                if TO(ci) = 0 then increment TO(ci) by 1
                end
        for p ∈ chld(n) do
            if CI(p) ∉ C(ci) then push CI(p) onto C(ci)
        end
    if TO(ci) = 0 then push ci onto Q
    end
O ← ∅
while Q ≠ ∅ do
    begin
    NQ ← ∅
    for O∈ Q do
        for cj ∈ C(cj) do
            begin
            TO(cj) = TO(cj) -1
            if TO(cj) = 0 then push cj onto NQ
            end
    O ← O ∪ Q
    Q ← NQ
    end
return O, T
```

Search-forward(ci, aci):

```
for nci ∈ C(ci) do
    begin
    if T(nci) = "a" and N(nci) ≠ ∅ then
        begin
        for n ∈ N(nci) do CI(n) ← aci
        N(aci) ← N(nci) ∪ N(aci)
        N(nci) ← ∅
        C(aci) ← C(aci) {nci}
        Search-forward(nci, aci)
        end
    if T(nci) = "c" then C(aci) ← C(aci) ∪ {nci}
    end
return
```

Search-backward(ci, aci):

```
for nci ∈ A(ci) do
    begin
    if T(nci) = "a" and N(nci) ≠ ∅ then
        begin
        for n ∈ N(nci) do CI(n) ← aci
        N(aci) ← N(nci) ∪ N(aci)
        N(nci) ← ∅
        A(aci) ← A(aci) {nci}
        Search-backward(nci, aci)
        end
    if T(nci) = "c" then A(aci) ← A(aci) ∪ {nci}
    end
return
```

It should be noted that the decomposition of the network and the de-cycling of components could be done together prior to calculating admissible valuations in contrast to the algorithm above. If the de-cycling is done prior to the calculating VAs then if the network is uninterpretable, i.e. some component has no admissible valuation, unnecessary work might be done. However, if the de-cycling is done as a prelude to calculations it can be invoked only when necessary. For example, if an assertional dependency is added to the network then the decomposition and de-cycling should not change (except for the addition of the new dependency) and it is not necessary to recalculate them.

In the following sections the de-cycling, permutation and checking for admissible valuations will be explained.

4.2.2.2 Removing Cycles

In order to interpret a network **<N,D>** containing cycles, it is necessary to convert **<N,D>** into an acyclic network **<N′,D′>**. Given certain restrictions, admissible valuations of **<N′,D′>** will be admissible valuations of **<N,D>**. This process is analogous to that of converting cyclic constraint satisfaction problems (CSPs) into acyclic CSPs [R. Dechter and J. Pearl, 1987].

The removal of cycles is done by duplicating a single node n, henceforth called a *cut-point*, from each cycle and changing the dependencies attached to n, $D(n)$, so that all the dependencies that were attached to n are attached to its double nn. Any admissible valuation V of **<N′,D′>** that gives equivalent values for n and nn, i.e. V(n) = V(nn), will be an admissible valuation of **<N,D>** [see theorem in 4.2.3.1].

Finding cutsets, i.e. a set of cut-points[4], in directed graphs is a well known problem in graph theory. Although the problem of finding a minimal cutset for any graph is known to be an NP-complete [Karp 1972], a linear time algorithm exists for finding minimal cutsets in *reducible* flow graphs [Shamir 1977].

A flow graph is a directed graph with a single root node, i.e. a node from which all other nodes are reachable. A *reducible graph* is one in which nodes can be successively merged together with a node being replaced by its single antecedent or parent. If there is more than one antecedent then the node cannot be merged, and if there are two nodes with multiple ancestors that form a cycle then the graph is not reducible. Another characterisation of non-reducible graphs is that there is more than one spanning tree of a network, i.e. the set of edges traversed by a depth-first search are not guaranteed to be unique.

[4] Cutsets are sometimes defined as sets of cut edges - i.e. the cycles are cut by removing edges rather than removing or duplicating nodes.

This is the source of failure for polynomial algorithms attempting to find minimal cutsets, for although a cutset can be generated from any spanning tree in polynomial time (see below) it will not necessarily (in the logical sense) be minimal. Only if a single spanning tree exists is the cutset guaranteed to be minimal, and for arbitrary graphs of n nodes there can be up to n^{n-2} spanning trees (for a complete graph, due to Cayley [1857]).

The algorithm *Decycle* below is based on one presented in [Shamir 1977] for finding minimal cutsets in reducible networks. *Decycle* works by first traversing forward across all the dependencies in a network, treating each dependency d as a set of separate edges (a,c) where a ∈ *ants*(d) and c = *cons*(d). Each edge is marked, M(a,c) = 1, as it is traversed, and can either be:

- a *backward* edge - if c has already been visited, V(c) = 1, and is above a on the same branch of the spanning tree, as represented by the stack B; or
- a *dag* edge - if c has not been visited or c is not on the same branch as a or is below a on the same branch (In which case (a, c) is respectively a *tree*, *cross* or *forward* edge - this distinction is however unnecessary).

If an edge (a, c) is labelled as backward then a is also labelled as the *head* of a backward link - H(c) ← 1. When a branch is being closed by popping a node n off the stack B, n is added to the current cutset C if it is an *active* head, where a head n is active given a cutset C if it is possible to trace a path forward over spanning edges from n to m avoiding C and (m, n) is a backward edge.

As each simple cycle (having only one node appearing twice) contains only one back edge it is only necessary to remove at most one edge per cycle. If a cycle has already been cut through the removal of some node (i.e. it is no longer active) then it is not necessary to remove the edge. Note - the third line of the algorithm chooses an arbitrary starting point for the depth-first search.

Having found a cutset, the cut-points are duplicated and the nodes that were previously attached to each cut point are transferred to the new duplicate node. So if d ∈ D(n) then *cons*(d) is reset from n to n′. This is denoted by adding d to D(n′) and removing d from D(n). The network produced by making these modifications is returned along with the set of cut-

points identified with their duplicate nodes.

Decycle(<**N,D**>)

```
M ← 0, V ← 0
B ← ∅, C ← ∅
for arbitrary n do push n onto B
while B = do
    begin
    a ← top(B)
    if ∃ c ∈ cons(a). M(a,c) = 0
        then begin
            M(a,c) ← 1
            if V(c) = 0
                then push c onto B
                else if c ∈ B then H(c) ← 1
            V(c) ← 1
            end
        else begin
            if H(a) = 1 and Active(a, B, C, <N,D>)
                then push a into C
            pop B
            end
    end
D ← ∅
for c ∈ C do
    begin
    c´ = new node
    D(c´) ← D(c)                    where D(c) = dependencies
    D(c) ← ∅                        attached to c
    push c´ into N and (c, c´) into D
    end
return <N,D>, D
```

This algorithm has complexity O(|N| × |E|) where E is the number of distinct antecedent/consequent edges. This is because each head (of which there are potentially |N|) must be checked to see if it is active which is done by traversing dag edges (of which there are potentially |E|), stopping if a cut-point is encountered). By clever labelling of nodes it is possible to check for active heads in constant time, thereby producing a linear time algorithm for minimal cutsets for reducible graphs. However, as IDNs are not restricted to reducible graphs it is not possible to use a linear time algorithm to produce minimal cutsets. In this case, the minimality of cutsets is given up in favour of a linear time algorithm.

```
Active(a,B,C,<N,D>)

active ← nil
S ← {a}
for a ∈ S while active = nil do
    begin
    T ← ∅
    for c ∈ cons(a) while active = nil do
        if c = a
            then active ← true
            else unless c ∈ B or c ∈ C
                then push c into T
    S ← T
    end
return active

Permute(𝒱, N)

If N = ∅ then return {{}}
N´ = tail(N)
n = head(N)
Ds = Permute(𝒱, N´)
NDs = ∅
for v ∈ 𝒱 do
    for D ∈ Ds do
        NDs = {⟶ᵥn ∪ D} ∪ NDs
return NDs
```

4.2.2.3 Checking Admissibility

The basic task of checking that an admissible interpretation V of an acyclic network **<N,D>** is admissible for the corresponding cyclic network is easy, given a set of node pairings D. The network is admissible iff for every pair of nodes $(c, c') \in D$, $V(c) = V(c')$.

However, it may be necessary to check other properties of admissible valuations for some other reason. For example, JTMS interpretations should not only be admissible, i.e. the value of each node must be supported by the values of its antecedents, but also be grounded. This is equivalent to saying that a node's value should not be self-supporting.

4.2.3 Termination and Complexity

Termination is easy to prove. For a finite network there are a finite number of components that will be found by depth-first search. The upward and downward search to join acyclic components into larger acyclic components is also guaranteed to terminate as there as no cycles in the network of components and the search will eventually run out of components.

For each cyclic component the modified algorithm to find cutsets is also known to terminate, being based on depth-first search (on finite acyclic networks representing dag edges). The cutset produced will be finite and for a finite set of values, the number of possible starting valuations for the acyclic valuation construction function will also be finite. Finally, as shown in §4.1.2 the acyclic valuation construction is also guaranteed to terminate.

The complexity of the complete interpretation algorithm is also easy to calculate as this has already been done for most of the elements of the total algorithm.

The complexity of the depth-first search is of the order max(|**N**|, |**D**|) which is the same for grouping the acyclic components and ordering them all. For a given cyclic component $\langle \mathbf{N}', \mathbf{D}' \rangle$, the complexity of finding a cutset is $O(|\mathbf{N}'| \times |\mathbf{D}|')$ and the complexity of calculating an admissible valuation is $O(|\mathbf{N}'| \times O(V^{\mathbf{N}}(n, V)))$ (for either a cyclic or acyclic component). However, the dominant factor in complexity is the number of admissible valuations that have to be calculated: for a cutset $C_{\langle \mathbf{N}', \mathbf{D}' \rangle}$ there are $|\mathcal{V}|^{|C_{\langle \mathbf{N}', \mathbf{D}' \rangle}|}$ possible initial valuations. If there are no cyclic components or each one contains just a single cycle then the complexity is polynomial (given a polynomial order set of summation functions) - otherwise the complexity is exponential in the size of the cutsets.

4.2.4 Soundness and Completeness of *Find-VAs*

To prove the soundness and completeness of *Find-VAs* it must be shown that the admissible valuations of de-cycled (previously cyclic) components are admissible valuations of the corresponding cyclic components (i.e. the output of *Find-Cyclic-VAs*), and that the union of the admissible valuations for each component is an admissible valuation of the whole network. To do this, I assume the correctness of *Decompose*, *Decycle, and Permute* (being either obvious or straight forward modifications to existing procedures) with the correctness

of *Find-Acyclic-VAs* proved in §4.1.3 and §4.1.4.

4.2.4.1 Soundness and Completeness of *Find-Cyclic-VAs*

To prove that *Find-VAs* is sound it must be shown that admissible valuations of acyclic components (resulting from the cutting of cyclic components) are admissible valuations of the original components. To prove that *Find-VAs* is complete it must be shown that each admissible valuation of a cyclic component produces a unique valuation over a cutset of that component, as it is the initial valuation of the cutset that is used to calculate the valuation for the whole component.

Defn A *gluing function* g is a function mapping a set of nodes to a subset of those nodes, $g{:}\mathbf{N}' \longrightarrow \mathbf{N}$, $\mathbf{N}' \subseteq \mathbf{N}$ s.t. $\forall n \in \mathbf{N}'$, $g(n) = n$.
If $g(n) = g(m)$ for $n \neq m$ then we say that n has been glued to m.

Defn $\langle \mathbf{N}', \mathbf{D}' \rangle$ is a *gluing* of $\langle \mathbf{N}, \mathbf{D} \rangle$ iff $\exists$ a gluing function $g{:}\mathbf{N}' \longrightarrow \mathbf{N}$ s.t.
$\forall d \in \mathcal{D}$, $d \in D \Leftrightarrow g(d) \in \mathbf{N}'$.

Defn $\langle \mathbf{N}', \mathbf{D}' \rangle$ is a *cutting* of $\langle \mathbf{N}, \mathbf{D} \rangle$ at n iff $\mathbf{N}' = \mathbf{N} \cup \{n'\}$ where $n' \notin \mathbf{N}$ and
$\mathbf{D}' = (\mathbf{D} \setminus D(n)) \cup \{ A_1, \ldots, A_p \longrightarrow_t n' \mid A_1, \ldots, A_p \longrightarrow_t n \in D \}$.
The function $g{:}\mathbf{N}' \longrightarrow \mathbf{N}$ is a gluing to recover $\langle \mathbf{N}, \mathbf{D} \rangle$ from $\langle \mathbf{N}, \mathbf{D} \bigcup D \rangle$.

Thm If $\langle \mathbf{N}', \mathbf{D}' \rangle$ is a cutting of $\langle \mathbf{N}, \mathbf{D} \rangle$ at c with gluing function
$g{:}\mathbf{N}' \longrightarrow \mathbf{N}$ s.t. $g^{-1}(c) \in \{c, c'\}$
and $V \in VA(\langle \mathbf{N}', \mathbf{D}' \cup \mathbf{D}_{ac} \rangle)$ s.t. $V(c) = V(c')$ where $\mathbf{D}_{ac} \subseteq \{ \longrightarrow_v c \mid v \in \mathcal{V} \}$,
then $V \mid_{\mathbf{N}} \in VA(\langle \mathbf{N}, \mathbf{D} \rangle)$.

Proof $\forall n \in \mathbf{N}$, $n \neq c \Rightarrow D(n) = D(g^{-1}(n))$ so obviously $D(n) \approx_g D(g^{-1}(n))$ and as
$\forall n \in \mathbf{N}$, $n \neq c \Rightarrow V \mid_{\mathbf{N}}(n) = V(g^{-1}(n))$ and $V \mid_{\mathbf{N}}(c) = V(c) = V(c')$ then
$V^{\mathbf{N}}(n, V \mid_{\mathbf{N}}) = V^{\mathbf{N}}(g^{-1}(n), V)$ by the Isomorphic Parents Lemma.
Given V is admissible (for $\langle \mathbf{N}', \mathbf{D}' \rangle$),
$V^{\mathbf{N}}(g^{-1}(n), V) = V(g^{-1}(n)) = V \mid_{\mathbf{N}}(n)$ as $g^{-1}(n) = n$.
For $n = c$, $D(c) \approx D(c')$ and $V^{\mathbf{N}}(c, V \mid_{\mathbf{N}}) = V^{\mathbf{N}}(c', V) = V(c') = V(c) = V \mid_{\mathbf{N}}(c)$.
So $\forall n \in \mathbf{N}$, $V^{\mathbf{N}}(n, V \mid_{\mathbf{N}}) = V \mid_{\mathbf{N}}(n)$ and $V \mid_{\mathbf{N}}$ is an admissible valuation of $\langle \mathbf{N}, \mathbf{D} \rangle$.
□

Cor If $\langle \mathbf{N}', \mathbf{D}' \rangle$ is a cutting of $\langle \mathbf{N}, \mathbf{D} \rangle$ at a set of points C with a gluing function $g: \mathbf{N}' \longrightarrow \mathbf{N}$ s.t. $\forall c \in C, g^{-1}(c) \in \{c, c'\}$ and $V \in VA(\langle \mathbf{N}', \mathbf{D}' \cup \mathbf{D}_{ac} \rangle$ s.t. $V(c) = V(c')$ where $\mathbf{D}_{ac} \subseteq \{ \longrightarrow_v c \mid c \in C, v \in \mathcal{V} \}$, then $V|_{\mathbf{N}} \in VA(\langle \mathbf{N}, \mathbf{D} \rangle)$.

Proof By repeated application of the theorem.

□

This corollary proves the soundness of *Find-Cyclic-VAs*.

Thm If C is a cut-set of $\langle \mathbf{N}, \mathbf{D} \rangle$ then $\forall V_1, V_2 \in VA(\langle \mathbf{N}, \mathbf{D} \rangle)$, $V_1 = V_2 \Leftrightarrow \forall c \in C, V_1(c) = V_2(c)$.

Proof

$\Rightarrow$ Trivially. $V_1 = V_2 \Rightarrow \forall n \in \mathbf{N}, V_1(n) = V_2(n) \Rightarrow \forall c \in C, V_1(c) = V_2(c)$.

$\Leftarrow$ Let $N_d = \{ n \in \mathbf{N} \mid V_1(n) \neq V_2(n) \}$.

Given $\forall c \in C, V_1(c) = V_2(c)$ then $N_d \cap C = \emptyset$ and N_d contains no cycles.

Therefore, assuming $N_d \neq \emptyset$, $\exists n \in N_d$ s.t. $pars(n) \cap N_d = \emptyset$ **

i.e. $\forall m \in pars(n), V_1(m) = V_2(m)$.

As V_1 and V_2 are admissible, $V_1(n) = V^{\mathbf{N}}(n, V_1)$ and $V_2(n) = V^{\mathbf{N}}(n, V_2)$.

Thus, by the Isomorphic Parent Lemma using the identity function as the isomorphism, $V^{\mathbf{N}}(n, V_1) = V^{\mathbf{N}}(n, V_2)$ and therefore $V_1(n) = V_2(n)$ and $n \notin N_d$ contradicting **.

Therefore N_d is empty and $V_1 = V_2$.

□

This proves the completeness of *Find-Cyclic-VAs*. Having cut $\langle \mathbf{N}, \mathbf{D} \rangle$ to produce $\langle \mathbf{N}', \mathbf{D}' \rangle$, each guess generated by *Permute* will produce an unique admissible valuation (using *Find-Acyclic-VAs*) of $\langle \mathbf{N}', \mathbf{D}' \rangle$. This valuation will also be an admissible valuation of $\langle \mathbf{N}, \mathbf{D} \rangle$ if it is consistent over the cut-points (which is checked by *Admissible*).

4.2.4.2 Soundness and Completeness of *Find-VAs* (again)

Having proved that each procedure is sound and complete, I now show that the union of valuations of the components of a network results in an admissible valuation of that network.

Defn A function $g:\mathbf{N'}_1 \cup \mathbf{N'}_2 \longrightarrow \mathbf{N}$ is a *composition function from* $\langle \mathbf{N}_1, \mathbf{D}_1 \rangle$ *and* $\langle \mathbf{N}_2, \mathbf{D}_2 \rangle$ *to* $\langle \mathbf{N}, \mathbf{D} \rangle$ iff

i) $\mathbf{N'}_1 \subseteq \mathbf{N}_1$, $\mathbf{N'}_2 \subseteq \mathbf{N}_2$

ii) $g|_{\mathbf{N'}_1}$ and $g|_{\mathbf{N'}_2}$ are isomorphisms

iii) g is surjective [onto - covers **N**]

iv) If $G_1 = \{ n \in \mathbf{N}_1 \mid \exists m \in \mathbf{N}_2, g(n) = g(m) \}$ and

$G_2 = \{ n \in \mathbf{N}_2 \mid \exists m \in \mathbf{N}_1, g(n) = g(m) \}$

then $\left[G_1 \cup \left[\bigcup_{n \in G_1} desc(n) \right] \right] \cap \left[\bigcup_{n \in G_1} ancs(n) \right] = \varnothing$

and $\left[\bigcup_{n \in G_2} desc(n) \right] \cap \left[G_2 \cup \left[\bigcup_{n \in G_2} ancs(n) \right] \right] = \varnothing$.

G_1 and G_2 are called the *joining sets* of $\mathbf{N}_1$ and $\mathbf{N}_2$ respectively.

Defn Given a composition function g, $\langle \mathbf{N}, \mathbf{D} \rangle$ is a *downstream composition of* $\langle \mathbf{N}_2, \mathbf{D}_2 \rangle$ *with* $\langle \mathbf{N}_1, \mathbf{D}_1 \rangle$, written $\langle \mathbf{N}, \mathbf{D} \rangle = \langle \mathbf{N}_2, \mathbf{D}_2 \rangle +>_g \langle \mathbf{N}_1, \mathbf{D}_1 \rangle$ iff

$\forall n \in \mathbf{N}$, if $\exists m \in \mathbf{N}_1$, $p \in \mathbf{N}_2$ s.t. $g(m) = g(p) = n$ then $D(n) \simeq_g D(m)$ otherwise $D(n) \simeq_g D(g^{-1}(n))$.

Thm Given $\langle \mathbf{N}, \mathbf{D} \rangle = \langle \mathbf{N}_2, \mathbf{D}_2 \rangle +>_g \langle \mathbf{N}_1, \mathbf{D}_1 \rangle$ for composition function $g:\mathbf{N'}_1 \cup \mathbf{N'}_2 \longrightarrow \mathbf{N}$, and $V_1 \in VA(\langle \mathbf{N}_1, \mathbf{D}_1 \rangle)$ and $V_2 \in VA(\langle \mathbf{N}_2, \mathbf{D}_2 \rangle)$,

if $\forall n \in \mathbf{N'}_1, m \in \mathbf{N'}_2, g(n) = g(m) \Rightarrow V_1(n) = V_2(m)$ then $V \in VA(\langle \mathbf{N}, \mathbf{D} \rangle)$

where $V = \{ (g(n), V_1(n) \mid n \in \mathbf{N'}_1 \} \cup \{ (g(n), V_2(n) \mid n \in \mathbf{N'}_2 \}$.

Proof Given the statement of the theorem,

$\forall n \in \mathbf{N'}_1$, $V_1(n) = V(g(n))$ and $\forall n \in \mathbf{N'}_2$, $V_2(n) = V(g(n))$.

So, $\forall n \in \mathbf{N}$, if $\exists m \in \mathbf{N'}_1$ s.t. $g(m) = n$ then $V_1(m) = V(n)$ and

$\forall p \in pars(m)$, $p \in \mathbf{N'}_1$,[5] $g(p) \in pars(n)$ and $V_1(p) = V(g(p))$.

Given this, and as by definition $D(m) \simeq_g D(g(m))$, the Isomorphic Parents Lemma can

[5] otherwise $\exists d \in D(m)$ s.t. $g(d)$ is undefined, $D(n)$ are not isomorphic to $D(m)$, and g is not a composition function.

be applied, so $V^{\mathbf{N}}(m, V_1) = V^{\mathbf{N}}(n, V)$ and given V_1 is admissible,

$V(n) = V_1(m) = V^{\mathbf{N}}(m, V_1) = V^{\mathbf{N}}(n, V)$.

If $\nexists n' \in \mathbf{N}'_1$ s.t. $g(n') = n$ then $\exists m \in \mathbf{N}'_2$ s.t. $g(m) = n$ [as g is onto].

So, repeating the argument as above [replacing V_1 by V_2 and $\mathbf{N}'_1$ by $\mathbf{N}'_2$]

$V(n) = V_2(m) = V^{\mathbf{N}}(m, V_2) = V^{\mathbf{N}}(n, V)$ and $\forall n \in \mathbf{N}$, $V(n) = V^{\mathbf{N}}(n, V)$.

Therefore V is an admissible valuation of $\langle \mathbf{N}, \mathbf{D} \rangle$.

□

This result proves the correctness and completeness of *Find-VAs* as follows.

Decompose produces an ordered set of components $\langle \mathbf{N}_1, \mathbf{D}_1 \rangle$,.... $\langle \mathbf{N}_l, \mathbf{D}_l \rangle$ s.t. $\langle \mathbf{N}, \mathbf{D} \rangle = \langle \mathbf{N}_1, \mathbf{D}_1 \rangle +>_{g_1}$,.... $+>_{g_{l-1}} \langle \mathbf{N}_l, \mathbf{D}_l \rangle$. The valuations of $\langle \mathbf{N}, \mathbf{D} \rangle$ are produced incrementally using the partial valuations of $\langle \mathbf{N}, \mathbf{D} \rangle$ that cover the first *n* components, $VA_n = \{V_1, ..., V_m\}$. Each $V_i \mid_{\mathbf{N}_{n+1}}$ generates a set of assertions $\mathbf{D}_{n+1,i} \subseteq \mathcal{D}_a$ that are added to $\langle \mathbf{N}_{n+1}, \mathbf{D}_{n+1} \rangle$ so that each admissible valuation V_j of $\langle \mathbf{N}_{n+1}, \mathbf{D}_{n+1} \cup \mathbf{D}_{n+1,i} \rangle$ agrees with V_i over $[\bigcup_{i \in [1,n]} \mathbf{N}_i] \cap \mathbf{N}_{n+1}$. So by the theorem $V_i \cup V_j$ is admissible over the first n+1 components and by induction the total valuations produced on termination are admissible for $\langle \mathbf{N}, \mathbf{D} \rangle$.

As *Find-Acyclic-VAs* and *Find-Cyclic-VAs* are known to be complete, then the resulting set of valuations

$$VAs = \{ V_1 \cup ... \cup V_k \mid V_i \in VA(\langle \mathbf{N}_i, \mathbf{D}_i \cup \mathbf{D}'_{i-1} \rangle), \quad \mathbf{D}'_{i-1} = \{ \longrightarrow_v n \mid n \in \mathbf{N}_i \cap \mathbf{N}_{i-1}, v = V_{i-1}(n) \}, \mathbf{D}'_0 = \varnothing \}$$

is guaranteed to be complete.

4.2.5 Odd Loops and Uninterpretable Networks

A known problem with the JTMS is the existence of uninterpretable dependency networks. Such networks contain cycles that include an odd number of non-monotonic dependencies (i.e. dependencies having out-nodes) and have acquired the name of "odd loops"[6]. They are

[6] This problem does not arise in non-monotonic ATMSs because the approach used does not involve non-monotonic justifications, nor cycles in the network - for further details see §5.1.

analogous to the Liar's paradox, "This statement is false" or "P is true iff P is false". Correspondingly, a cycle containing an even number (>0) of non-monotonic dependencies are called "even loops".

Given a network < {n}, $\{\{n\}_{out} \longrightarrow_{SL} n\}$ >, if V(n) = *out*, then $V^{N}(n, V)$ = *in* and if V(n) = *in*, then $V^{N}(n, V)$ = *out*. No valuation of such a network is going to be admissible. When we cut the network to produce < {n, n´}, $\{\{n\}_{out} \longrightarrow_{SL} n'\}$ > the valuations produced do not give consistent results for n and n´.

By excluding networks that contain odd-loops one can make sure that all networks have at least one admissible valuation but Goodwin [1987] concludes that

> *"it is not natural to permit even loops and encourage the programmer to use them deliberately to code arbitrary choices, but to forbid the use of odd loops"*

for the use of even loops naturally leads to the creation of odd loops. The conclusion to be drawn from his work is that no non-monotonic dependencies should be allowed to occur in cycles.

This solution is not at all satisfactory from the point of view of the expressiveness of dependency networks - by Goodwin's own admission both odd and even loops are naturally occurring or intuitive structures. This point is highlighted in §5.2 on default logic where cycles are again seen as naturally occurring phenomena.

Furthermore, the presence of odd loops in a network is insufficient to make a network uninterpretable. For example < {n}, $\{\{n\}_{out} \longrightarrow_{SL} n, \longrightarrow_{SL} n\}$ >. As n has independent support, i.e. has a valid dependency other than the odd loop, the valuation V = {(n, *in*)} is admissible. Thus we can see that though a network may contain odd loops those loops may not prevent an admissible valuation from being constructed.

Odd loops are a problem for previous JTMS-style algorithms for two reasons: they may cause non-termination; or they introduce multiple solutions resulting in the random choice of just a single solution. By constructing admissible valuations through stratification, de-cycling and consistency checks on the resulting valuations, termination is guaranteed (and at the earliest possible time when no solutions exist) and multiple solutions obtained as required (or not as the case may be) with no restriction on the type of network allowed.

Finally, IDNs are presented as a general tool or technique. As such, any IDN system that is created may generate uninterpretable networks. However, it is not necessary to discover characteristic properties of such networks in advance. If any network is uninterpretable the algorithm presented will discover this. If this is the case then for every possible set of assertions added to the cut-points of the cycles, the resulting valuations will be inconsistent over some cut-point and its double, i.e. $V(c) \neq V(c')$. This demonstrates the flexibility of the IDN approach.

4.2.6 Modifications to the Interpretation of Cycles

The algorithm given above will calculate all possible admissible valuations of cycles. This may not be what is required. If cycles represent equivalences, then one might want to make a minimal commitment and guess *nil* values. This is the desired result in the J- and ATMSs where a self-supporting cycle is undesirable. For example, the network

$$< \{a\ b\ c\ d\},\ \{a_{in} \longrightarrow_{SL} b,\ c_{in} \longrightarrow_{SL} d,\ b_{in} \longrightarrow_{SL} d,\ d_{in} \longrightarrow_{SL} b\ \} >$$

has valuations

$V_1 = \{(a, out), (b, out), (c, out), (d, out)\}$

$V_2 = \{(a, out), (b, out), (c, in), (d, in)\}$

but only the first is *grounded* in the JTMS, i.e. has a non-circular set of dependencies supporting each value.

Different approaches may be more suitable in different cases. For the ATMS it is sufficient to choose minimal admissible valuations. This can be done by using a *nil* interpretation for all cut-nodes. The values assigned to the duplicate nodes can then be used as a second guess for the cut-nodes and the resulting admissible valuation of the acyclic component will also be admissible for the cyclic component.

In the case of the JTMS a node's value is not self-supporting if it is *nil*, or if it is part of an even non-monotonic cycle. This can be tested with a modified ATMS-type algorithm. Given a valuation V of root nodes (those with no antecedents) a modified valuation V′ is constructed where $V'(n) = \{\{n\}\}$ if $V(n) = in$ and $V'(n) = out$ if $V(n) = out$. The summation

function for +ve antecedents and for the node level (S^N) is as for normal ATMS dependencies (see §3.4.3) with the following additions and changes:

- any *out*s in a label L can be discarded unless the whole label consists of *out*s, in which case L ≡ *out*;
- any *in*s contained in an environment E can be discarded, unless all elements are *in*s, in which case E ≡ *in*;
- if there is an environment E_i equal to *in* in label L then L ≡ *in*;

Using the following set of summation functions for SL-dependencies:

$S^{SL}_{in}(v_1, ..., v_n) = out$ iff $\exists i.v_i = out$

$= S^{A}_{s,+ve}(v'_1, ..., v'_n)$ otherwise

where $v'_i = \{\{\}\}$ if $v_i = in$, $v'_i = v_i$ otherwise

$S^{SL}_{out}(v_1, ..., v_n) = in$ iff $\forall i.v_i = out$

$= out$ otherwise

$S^{SL}(v_1, v_2) = out$ iff $v_2 = out$

$= in$ iff $\{\} \in v_1$ and $v_2 = in$

$= v_1$ otherwise

each duplicate node d′ in the acyclic component will end up with one of the following values:

in representing a grounded value *in*;

out representing a grounded value *out*; or

an ATMS type label L.

In the last case, d′ has a grounded value of *in* iff there is an environment E ∈ L where all the nodes in E have a grounded value of *in*. This can be checked by recursively substituting *in* for cut-nodes whose duplicate nodes have a value *in*, bearing in mind the equivalence rules above. If a point arises where no more substitutions can be made and there are still ATMS labels assigned to nodes then the valuation is not grounded.

For example, consider the partial admissible valuation of a cut acyclic component:

$V(a') = in$, $V(b') = in$, $V(c') = \{\{a\}\{b\ d\}\}$, $V(d') = \{\{c\}\}$

Substituting *in* for a, $V(c') \equiv \{\{in\}\} \equiv \{in\} \equiv in$. Recursively substituting *in* for c, $V(d') \equiv in$ and the admissible valuation $V(a) = V(b) = V(c) = V(d) = in$ is grounded.

If $V(c')$ did not contain {a} then *in* could be substituted for b giving $V(c') = \{\{a\}\{in\ d\}\} \equiv \{\{a\}\{d\}\}$. However, no more substitutions can be made and therefore the admissible valuation is not grounded.

The need for a special purpose approach is even more appropriate when the set of values being dealt with is infinite, e.g. probabilities or certainty factors are real numbers in the range [0,1]. In such cases it is impossible to enumerate all possible guesses and admissible valuations. Even if the values were restricted to a certain precision so that the set of values becomes finite, i.e. 2 decimal places in the range [0, 1] gives 100 values, the computational cost of enumerating all such guesses and finding all admissible valuations would be computationally intractable without this approach.

In some cases the admissible valuations of cycles cannot be created using summation functions (e.g. for probabilistic TMSs [Falkenhainer 1987]). In such cases special purpose algorithms may be needed, or alternatively the cycle can be treated as a single node thus allowing an approximation to an admissible valuation to be calculated.

4.3 Methods of Reasoning

As stated in Chapter 1 this thesis is concerned with the process of changing one's view or interpretation of the world with respect to a given theory about the world. Whereas logic is primarily concerned with theoremhood or deriving relationships between propositions as embodied by queries in logic programming, IDNs are concerned with establishing consequences of a set of axioms - i.e. what follows or can be derived in a given situation. Above and beyond that, IDNs are concerned with the changing status of information as those axioms, premises or assumptions change.

Sections 4.1 and 4.2 have been concerned with the interpretation of networks without reference to any existing valuation, something that is required if IDNs are to be viewed as declarative knowledge representations divorced from programs that interpret them. This

process is akin to constructing an interpretation of a given theory and was the first of the theory questions on the list in §4.0.2 to be addressed. The following section looks at other ways of interacting with such a theory, given an existing interpretation.

4.3.1 Data-Driven Processing - Propagating Changes

Many reasoning tasks are incremental in nature and involve a dialogue between a system and the user of that system. In general, assumptions may have been made that are revoked at a later date, more information may have been asserted or previous information may become outdated. It is the need to maintain an up-to-date interpretation of information that led to the development of TMSs and it is this idea that forms the basis of IDNs.

If a system works in a changing environment or produces changes to its environment (i.e. information cannot just be accumulated and then have its status or "validity" remain unchanged) then the value attached to each datum must be continually supported by evidence to support that value. In classical logic for example, the emphasis is on deriving implicit information from explicit axioms but is less concerned with what happens when the explicit axioms change.

This can be clearly seen in the relationship between non-monotonic logics and TMSs. Take McDermott and Doyle's logic [1980] that uses a modal operator M to denote consistency where Mp is true iff ¬p is not true. Consider the classic bird example consisting of the theory where all birds fly unless it has been proved otherwise, and all penguins are birds and all penguins don't fly.

$$T = \{\ \forall x\ b(x) \wedge Mf(x) \rightarrow f(x),\ \forall x\ p(x) \rightarrow b(x),$$
$$\forall x\ p(x) \rightarrow \neg f(x)\}$$

Given Tweety is a bird, i.e. a set of facts $S = \{b(t)\}$, then the logic can be used to construct a minimal fixed point, $TH(S \cup T)$, representing the non-monotonic theorems that includes the "assumption" that it is consistent that Tweety flies Mf(t), and proposition that Tweety can in fact fly, f(t). If S is expanded to include the fact that Tweety is a penguin, $S' = \{b(t), p(t)\}$, then $TH(S' \cup T)$ contains neither Mf(t) nor f(t) due to the presence of ¬f(t) in the (monotonic) theorems of $S' \cup T$.

So the logic tells us what propositions are or are not theorems given some set of axioms, but does nothing to tell us how to derive a new fixed point from an old one. For this we need a mechanism like a TMS or an IDN to record derivations and update the validity or truth value of propositions as the validity of its derivation changes.

In previous sections the construction of interpretations was examined - in this section the process of updating interpretations given changes in the underlying network is examined.

There are four ways of characterising changes to a (IDN) dependency network: additions or deletions of normal or assertional dependencies. Changes to assertional dependencies indicate changes to the known or asserted facts or beliefs, while changes to normal dependencies indicate a change in the structure of an underlying theory. The distinction between changes in assertions and changes in normal dependencies is important for the latter may lead to changes in the decomposition of the network while the former is guaranteed to leave the decomposition intact.

It is the computational expense of calculating cutsets for cyclic components, along with the cost of constructing valuations from guesses of cut-point values, that dominates the complexity of finding valuations of cyclic networks. Therefore, changes to the network that leave the cutsets unchanged result in updates of at least half the cost of those that do not. Assertional updates fall into this category and it is only the addition or deletion of dependencies that adds new links between different components that necessitate the recalculation of the decomposition, and only those that add new cycles that necessitate the calculation of new cutsets. This only occurs when a dependency is added that links a previously downstream component to an upstream one.

Given a recalculation of the decomposition and cutsets as necessary, the actual interpretation must be updated. There are two obvious ways to approach this. The first is to recalculate admissible valuations of the subnetwork consisting of all the descendents of the node whose attached dependencies have changed. The second is to only recalculate the values of nodes where it is known that at least one antecedent node's value has changed.

The first is a simple approach, known to be exponential in the number of cut-points and the size of any cycles, and linear in the size of acyclic components. However, values do not

have to be calculated at each node as no change may have occurred in the parent nodes' values. When calculating V′ from V, if V′(n) = V(n) for all n ∈ *ants*(m) then V′(m) = V(m). This approach is guaranteed to be correct by the gluing theorem (4.2.4.1). The second is attractive in offering a chance to reduce the complexity, but must be approached in the correct way for otherwise the worst case complexity can be increased by introducing the opportunity to do unnecessary recalculations.

4.3.1.1 Incremental Updates

The easiest way to incrementally update a valuation of an acyclic network is to maintain a list of nodes or an agenda[7] where one of that node's antecedent's value has changed and only recalculate values for nodes on this list. Having taken a node from the front of the agenda and recalculated the value, if the value is different from the previous value then the consequents of the node should be placed on the agenda. When the agenda is empty then there are no more values to recalculate and all the necessary values have been recalculated to produce a new admissible valuation. However, it is important that the agenda is ordered in the correct way. For example, consider the network below and the addition of $\longrightarrow_{SL}$ to a.

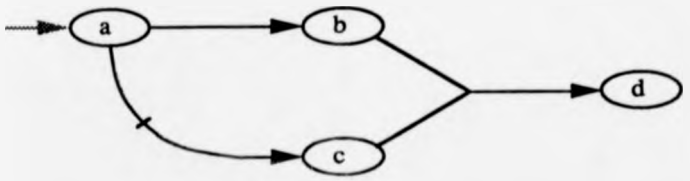


Figure 4.3.1.1 (a)

If the updating is done in a depth-first manner with new nodes added to the front of the agenda we could get the following:

[7] to use the AI flavoured synonym of queue

valuation				
a	b	c	d	agenda
out	*out*	*in*	*out*	a
in	*out*	*in*	*out*	b c
in	*in*	*in*	*out*	d c
in	*in*	*in*	*in*	c
in	*in*	*out*	*in*	d
in	*in*	*out*	*out*	-

The unnecessary changing of d from *out* to *in* and back to *out* could cause a ripple of unnecessary changes to propagate throughout the descendents of d (if any exist). Similarly, networks exist that produce similar behaviour when using breadth-first recalculation. It is only by ordering the recalculations according to the depth of the nodes that this problem is avoided, thus making sure that in the example, the value for b is calculated before that of d.

The updating of cycles is not as efficient compared to acyclic networks, as the normal cycle of guessing, construction and checking valuations must be carried out. However, the previous valuation provides a good initial guess so that if updating terminates half-way down the acyclic dual then the resulting valuation is admissible. This then gives us a useful heuristic for calculating admissible valuations for cycles. If a guess results in an inconsistent match between cut-nodes then use the resulting valuation as the new guess, if it has not been used before. This suggests another way of constructing admissible valuations for acyclic networks. Rather than recording when all a node's antecedents have been calculated, and then calculating the node's value, all the nodes could be ordered according to their depths and then be calculated in that order.

algorithm

label nodes with depths (or update labelling)

initialise agenda

loop: until empty agenda

remove first node from agenda

recalculate value of node

compare old and new values

if values different add consequents and reorder agenda

This method compares with those of Doyle [1979] and Goodwin [1987] which also determine the set of nodes, the *disturbance set*, whose values might change, and only recalculate values for those nodes. Their methods rely on *supporting justifications* and *supporting nodes*. These are the dependencies and nodes that currently support the node's value and only if their values change will that of the supported node change. So when an addition or deletion is made to a node's set of attached dependencies, $D(n)$, the *affected consequents* are those consequents of the node which have n as a supporting node and the disturbance list is just the transitive closure of the affected consequences.

The method presented here is different in two respects: it does not use the concept of supporting nodes and justifications; and it calculates the disturbance set iteratively in the course of creating the new valuation. The first means that it is (potentially) less efficient for summation functions where not all of the attached dependencies are needed to support a particular node. This results in unnecessary calculations being done when the changed antecedents don't belong to the supporting nodes. However this may lead to nodes being included in the disturbance set when alternative supporting justifications exist and their values will not actually change[8]. This coupled with the fact that the disturbance set is not calculated iteratively in the JTMS means that many nodes will be included in the disturbance set when they should not have been.

[8] Note that although the values may not actually change it *is* necessary to update the supporting nodes and justifications.

4.3.2 Goal- or Query-Driven Processing

The construction or updating of admissible valuations are essentially data-driven operations in that values calculated for upstream or antecedent nodes are used to calculate values for downstream nodes or consequents. There is a range of behaviour that occurs in AI systems that is the opposite: processing starts at some consequent and proceeds backwards attempting to find or allocate values to antecedents. Examples of this backtracking behaviour include abduction in logic, backward chaining expert systems, and contradiction resolution in TMSs, be it explicit as in the JTMS's dependency directed backtracking or implicit in the case of the ATMS's "no-goods".

In this section I shall look at some existing work in explanation and backtracking and attempt to show how backtracking algorithms can vary and what information is needed to implement such algorithms. This will facilitate the design of backtracking algorithms but in itself this section cannot give specific algorithms to solve specific problems.

Abductive reasoning is characterised by the following type of reasoning: $\alpha \rightarrow \beta, \beta \models \alpha$. This is unsound in classical logic but has an intuitive appeal in the everyday world. This approach to reasoning has been much studied in AI or more specifically, non-monotonic reasoning. In particular Reiter and de Kleer [1987] present a method for providing minimal explanations for propositional clauses (consisting of a disjunction of ground literals) while Poole [1989] looks at explanations for observations in a default system and the different results produced by minimising the number of assumptions, abnormalities and/or implications.

Forward chaining expert systems work by matching facts in the working memory with the antecedents or left hand side of rules, as modelled by the addition of assertion dependencies and updating valuations in IDNs (see Chapter 6 for more details). Backward chaining systems match goals or queries with the right hand side of rules, and then use the left hand side of such rules as new subgoals. Processing proceeds until either subgoals match with known facts or designated subgoals, or facts are reached where questions are asked to obtain such information.

Finally, in contradiction resolution, when a contradiction is detected, the source(s) of that contradiction must be identified so that the derivation of the contradictory information is blocked. In the JTMS this is done by explicitly traversing up the network to find non-monotonic links that support the contradiction, while in the ATMS the graph is not actually traversed as each node's label records the assumptions underlying its derivation, but these nodes should form the starting points for the removal of no-good environments from each node's label.

As the examples show, most backtracking algorithms could or should be followed by a period of forward processing. Having obtained a possible explanation or set of propositions supporting observed phenomena, the explanation should be asserted to see what other consequences follow, highlighting further information that could confirm or disconfirm the hypothetical explanation. So if Tweety is a non-flying bird then Tweety is an abnormal bird with respect to (the general property of) flying[9]. Looking for possible sources of confirmation of this (independent of Tweety's inability to fly) might generate the possible explanations that Tweety is a penguin and/or Tweety has a broken wing and/or Tweety's feet are set in concrete! Asserting that Tweety is a penguin leads to support for Tweety having flippers and if this generates a contradiction at a later date then the assumption of Tweety being a penguin will be revoked.

In the JTMS, having found a possible culprit responsible for a contradiction, that assumption is retracted, and the valuation updated to see if the contradiction is removed. Failure to achieve this results in another culprit being selected. The persistence of contradictions after backtracking is the result of three factors: the recording of only one of possibly many supporting justifications means that valid but previously hidden justifications may still support a contradiction; the process of checking that culprits are actually responsible for the contradiction is an approximate one[10]; but more seriously, the removal of

[9] This statement of abnormality is with respect to a certain property and relies on the fact that the property in question is a property that applies to the larger class. E.g. birds fly, Tweety is a bird that doesn't fly, so Tweety is abnormal w.r.t. the class of birds that do fly, i.e. w.r.t to flying. It is not a statement that Tweety is abnormal because he flies (for he doesn't!).

[10] Both these are implementation issues and can be resolved by adopting the IDN approach of "summing support".

a culprit assumption may make previously invalid justifications valid. This problem can only be addressed by looking at entailment relations rather than working (i.e. backtracking) within a particular interpretation, although more sophisticated backtracking algorithms can remove some such cases.

4.3.2.1 Explanations and Backtracking

In terms of IDNs, backtracking corresponds to having a network **<N,D>** (theory) with an admissible valuation V (consistent interpretation) and a partial valuation V_O over some set of nodes (observations or desired goals) and looking for a set of assertional dependencies $D \subseteq D_a$ (an explanation) s.t. $\exists\ V' \in VA(\mathbf{<N,D \cup D>})$ and $V_O \subseteq V'$. In addition, conditions relating V to V′ may have an impact on which D and V′ are chosen if there are multiple solutions. For example, a minimal D might be desired, or a minimal difference between V and V′ preferred.

Jackson [1989] and Poole [1989] both give criteria for preferring one explanation to another. They both include *minimality* with explanation E_1 preferred to E_2 if $E_1 \subseteq E_2$ where an explanation for Jackson is just some set of clauses whilst for Poole an explanation is a set of ground instances of some set of designated formulae or *hypotheses*.

Additionally Jackson does not allow trivial explanations where some observations are entailed directly by the explanation itself. I.e. the explanation of β with regard to the theory $\{\alpha \rightarrow \beta\}$ as β or alternatively $\{\gamma, \gamma \rightarrow \beta\}$ is trivial - one must use some of the information in the theory to derive the observation.

This problem is avoided by Poole by choosing explanations from a set of hypotheses, though if the observation *o* to be explained were one of those hypotheses, then {o} would be a perfectly correct explanation. Poole adds an extra criteria: that of choosing the *least presumptive* explanations, where E_1 is less presumptive than E_2 if, given theory T, $T \cup E_2 \models E_1$.

We can view these criteria as pointers on where and how to add D to **<N,D>** to explain V_O. If we look at the subnetwork **<N′,D′>** supporting the nodes $O \subseteq N$ in V_O, then the first (minimality) merely prevents the spread of the explanation unnecessarily across **<N′,D′>**

while the rest point to the level at which the explanation should be pitched. The triviality condition says that the dependencies in D should not be attached directly to O while presumption says that they should be attached as near as possible.

Poole explicitly rules out the possibility of choosing explanations based on the number of assumptions though his reason[11] is unclear. Given William of Occam's famous maxim this would seem to be an excellent criterion to use. In another guise, Occam's razor supports the principle of having the explanation as close to the observation as possible, for if $T \cup E_2 \models E_1$ then $\mathit{Thms}(T \cup E_2) \supseteq \mathit{Thms}(T \cup E_1)$ (given a monotonic entailment) and a choice of E_2 over E_1 would be adding unnecessarily to the derived information. However, there are reasons for choosing to add D at a higher level, further away from O. If root nodes represent physical events while their descendents are concepts or inferred information then explanations at a higher level in the network provide more concrete explanations. Alternatively, in a data-driven application when more information is required to progress towards a solution, backtracking to a higher level will produce more information.

The choice of when to backtrack and where to backtrack to is a complex decision to which I believe no simple rule can be applied. Consider the network:

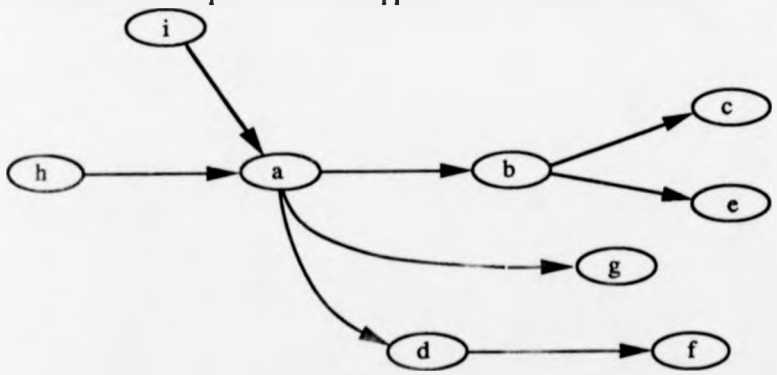


Figure 4.3.2.1 (a)

[11] He states that "such comparators have too many problems of slight changes to the representation of the problem domain giving different answers" but his example, i.e. "it is not reasonable to always prefer one rare disease over two common diseases" introduces a whole new dimension, e.g. probabilities, into the discussion without clarifying his previous statement.

Given a desire to support c and f, should backtracking stop at {b, d} (the closest explanation, minimal disturbance) or at {a} (minimal explanation) or at {h, i} (highest level)?

4.3.2.2 Finding Support

Having decided the criteria for determining at what level backtracking should stop, it is necessary to determine which dependencies and nodes should be backtracked over before reaching the desired level. For example, if the backtracking were an attempt to find which nodes were supporting an *in* value for a JTMS-IDN, then the backtracking should be over dependencies with *in* values. If the backtracking were an attempt to find a set of nodes to be asserted so as to change a node's value from *out* to *in* then it would be necessary to backtrack over dependencies with *out* values.

In order to guide the backtracking it is necessary to know:

- what consequent value is desired or currently supported; and
- what antecedent values are supporting the consequent (or are needed to support the desired consequent value).

In order to do the latter it is necessary to have some model of how a particular dependency works (i.e. have a model of the summation functions) so as to drive the dependency backward. For example, for a JTMS-IDN node to have the value *in* it is necessary that one or more dependency has the value *in*. An SL-dependency only has a value *in* iff all in-nodes are *in* and all out-nodes are *out*.

Constructing the inverse of a dependency's summation functions introduces one of the main sources of problems in backtracking. A dependency's value is a function of the values of the antecedents but this function is typically many-to-one. This means the inverse is not a well defined function and any number of valuations for the antecedents may produce the required value. This means that there are many different possible explanations, or possible sets of supporting nodes. Managing this is one of the main problems for any backtracking algorithm.

4.3.2.3 Entailment Relations

In query-driven processing, a simple traverse of the graph can provide information about approximate support relations. Though this may be sufficient to direct the gathering of information in the problem domain, this is not sufficient in the case of direct queries or counterfactual reasoning of the form "does Γ entail Δ?" or "if V(n) = v would V(m) = w?". In the same way that logic develops syntactic entailment $\vdash$ via a proof theory so as to eliminate the need to consider semantic entailment at the model theory level, so I would like to relate the entailment defined in Chapter 3 to the structure of a network.

Unfortunately this is not possible for general IDN systems. Given a set dependencies $D(n) = d_1, ..., d_p$ and valuation V s.t. $V^{\mathbf{N}}(d, V) = v$, unless $S^{\mathbf{N}}$ is fixed for V, i.e.

$$\forall\, w \in \mathcal{V}.\ S^{\mathbf{N}}(v_1, ..., v_r) = S^{\mathbf{N}}(v_1, ..., v_r, w)$$

then the addition of the assertional dependency $\longrightarrow_w n$ would invalidate $V \mid_P \models (n,v)$[12], where P is the set of parents of n.

This does not prevent us developing the relationship between $\models$ and $\longrightarrow$ in certain systems. Such a class of systems are those on whose values $\mathcal{V}$ an order $\leq$ can be defined. This enables us to define an ordered entailment $_{\leq}\models$ where $\Gamma\ _{\leq}\models \Delta$ iff any model satisfying Γ satisfies Δ to the required degree or more so. Thus the interpretation function i for a node/value pair is changed from that in §4.2.2.1 to

Defn $i(\{\langle \mathbf{N},\mathbf{D}\rangle, V\}, (n, v))$ = "true" iff $V(n) \geq v$
= "false" otherwise

Ordered entailment is defined as before using this new definition for i and produces the same results except in those entailments that use node/value pairs. In particular, writing $M(\langle \mathbf{N},\mathbf{D}\rangle)$ for the admissible valuations of $\langle \mathbf{N},\mathbf{D}\rangle$ and its extensions $E(\langle \mathbf{N},\mathbf{D}\rangle)$, i.e.

$$M(\langle \mathbf{N},\mathbf{D}\rangle) = \{\, V \mid \exists\, \{\langle \mathbf{N}',\mathbf{D}'\rangle, V\} \in C.\ \langle \mathbf{N}',\mathbf{D}'\rangle \in E(\langle \mathbf{N},\mathbf{D}\rangle)\}$$
$$= \bigcup_{\langle \mathbf{N}',\mathbf{D}'\rangle \in E(\langle \mathbf{N},\mathbf{D}\rangle)} VA(\langle \mathbf{N}',\mathbf{D}'\rangle)$$

we get the following corollary:

[12] assuming it is consistent to add $\longrightarrow_w$

Cor For $\Gamma = \{ (n_1, v_1), ..., (n_p, v_p) \}$, $\Delta = \{ (m_1, w_1), ..., (m_q, w_q) \}$,

$\Gamma \mid\!\!\sim_{\leq} \Delta$ iff $\forall V \in M(\langle \mathbf{N}, \mathbf{D} \rangle)$. $(\forall i \in [1,p].\ V(n_i) = v_i) \Rightarrow (\forall j \in [1,q].\ V(m_j) \geq w_j)$

However, the definition of ordered entailment is too weak to guarantee a correspondence between dependencies and entailment. If both negative and positive support are allowed it is not possible to have $(n, v) \mid\!\!\sim_{\leq} (m, w)$ purely on the basis of a dependency linking n and m - it may be possible to add negative assertional dependencies that lower the supported value of m, thus negating the right hand side of the theorem.

In general it is difficult to provide a general notion of entailment (based on the semantic structures outlined in §3.3) that means that α entails β iff there is a certain type of dependency (or set of dependencies) linking α and β. It may be possible to do this on an IDN by IDN basis for those that avoid certain properties, i.e. allowing dependencies to decrease support or having more than two values, without using any notion of ordered entailment.

4.3.2.4 Implementing the JTMS as an IDN

In §3.4.2 the JTMS was considered as an IDN but only one of the three basic JTMS procedures (Truth Maintenance) was covered in that section. Finding supporting nodes for contradictions (through dependency directed backtracking) and the interpretation of CP-justifications were both put off on the grounds that they are meta-level inference procedures.

In this last section these types (backtracking and checking entailment relationships) of algorithm have been described in general terms without reference to how those algorithms could be implemented for a JTMS. In this section I would like to sketch the network conditions that determine: which nodes should be considered as culprits for a contradiction; and under what conditions a CP-justification is valid.

A node n is a culprit (or underlying assumption) of a contradiction (given an interpretation V) if it is *in*, and changing its value to *out* would remove one source of support for a contradiction. This is the case iff there is a path of dependencies $\{d_1, ..., d_m\}$ s.t.:

(C1) $\forall i. V^{\mathbf{D}}(d_i, V) = in$;

(C2) $a \in ants_{+}(d_1)$;

(C3) $\forall i<m.\ cons(d_i) \in ants_+(d_{i+1})$; and

(C4) $cons(d_m) = n$;

where $ants_+$ are the positive SL-antecedents.

The entailment relationship between the a_is and the contradiction c is embodied by the CP-justification

$$c_{cons}\ \{a_1, ..., a_k\}_{in-hyp}\ \{b_1, ..., b_l\}_{out-hyp} \longrightarrow_{CP} d$$

This justification is valid (has a value *in*) iff there is a set of dependencies $D = \{d_1, ..., d_m\}$ s.t.:

(CP1) $\forall a_i.\ \exists \{d'_1, ..., d'_m\} \subseteq D$ satisfying C2-C4 above (substituting a_i for a and c for n);

(CP2) $\forall b_j.\ \exists \{d'_1, ..., d'_m\} \subseteq D$ satisfying C3 and C4 above and where $b_j \in ants_-(d_1)$ and $cons(d_1) \in ants_+(d_2)$; and

(CP3) $\forall d_i.\ \forall a \in ants(d_i).\ (a = a_i \text{ or } a = b_j)$ or $(\exists d_j.\ cons(d_j) = a)$ or
$(\forall V \in VA(\langle \mathbf{N}, \mathbf{D} \rangle).\ (a \in ants+(d_i) \Rightarrow V(a) = in) \wedge (a \in ants-(d_i) \Rightarrow V(a) = out))$.

In other words, the CP-justification is valid iff for all in-hypotheses there is a path of SL-dependencies linked through the in-nodes, and for the out-hypotheses there is a path of SL-dependencies similarly linked (barring the first dependency where the out-hypothesis is an out-node). Furthermore, all the antecedents for the set of dependencies D must either be a hypothesis of the CP-justification, the consequent of one of the dependencies in D, or must have a constant value over all possible interpretations of the network.

These criteria can be used to specify procedures to perform DDB and to check the validity of CP-justifications. The procedures can either be explicitly called after the construction of admissible valuations as necessary (e.g. if a contradiction node is *in*) or by the use of consumers (see §6.5.1).

4.4 Conclusion

The goal of this chapter has been to look at how the information contained in an IDN could be used. Various interactions are proposed in §4.0.2 and of these the construction of an interpretation of the network (representing a closure of possible inferences) is the most

important. The main contribution of this chapter is to give an algorithm for constructing the admissible valuations of both acyclic (§4.1) and cyclic (§4.2) IDN networks. The rest of the interactions proposed in §4.0.2 are covered in §4.3: incremental changes to the network structure are covered in §4.3.1 while the generation of explanations and tracing support for values are covered in §4.3.2.

Much of the work in the last two sections can only be described in general outlines, through sketching possible approaches or algorithms. This is necessary due to the complexity of the problems and the wide range of possible IDNs that any general algorithm would have to cover. In order to implement such reasoning for specific IDN systems it would be necessary to fill in the specific detail. The IDN framework provides a vocabulary for describing the problems but is too general to allow for detailed solutions.

CHAPTER 5

Applications of IDNs

In preceding chapters an analysis of existing approaches to belief revision based on Truth Maintenance Systems has been used to motivate a general framework for managing a justification-based approach to belief revision. In this chapter some existing knowledge representation paradigms will be couched in terms of Interpreted Dependency Networks, thus providing an insight into their operation and semantics and giving rise to new methods of interaction.

These examples provide a justification for an IDN-based approach to belief revision, showing the scope of existing representations and problems that can be formulated as IDNs. Many of the examples also contain a contribution towards solving or understanding the original problem and/or domain.

5.1 Non-Monotonic ATMSs

In §2.1.2 it was shown how the ATMS could be used to encode non-monotonic justifications of the form "If π is believed and ψ is not believed then believe ρ". Both de Kleer [1986b] and Dressler [1987] resort to adding extra constructs outside the normal label propagation algorithm. In this section I propose a scheme whereby non-monotonic justifications can be handled within a normal ATMS label propagation scheme. It is only necessary to change the set of labels allowed and alter the summation functions responsible for the calculation of labels.

The resulting non-monotonic ATMS (NATMS) has the benefits of being more expressive than a standard ATMS and more efficient than existing extended ATMSs (EATMS) by keeping a small no-good database and eliminating the need for an intermediate phase of environment expansion in order to generate a context from an environment. This highlights the flexibility of IDNs in two different ways. First, the IDN framework allows for the easy specification of TMSs. Second, by changing the interpretation mechanism it is possible to switch easily from JTMS to ATMS-style Truth Maintenance over the same dependency network.

5.1.1 Specification

This approach to NATMSs was inspired by the original multiple context TMS [Martins and Shapiro 1983] in which contradictions were not stored separately as no-goods but were represented as restrictions on the derivation of nodes. Given a justification $n_{out} \longrightarrow_{SL} m$, m should be in any context in which n is not derived. So given a label for n, $V(n) = \{E_1 \ldots E_r\}$, if an environment E doesn't contain any E_i as a complete subset then n should not be in E's context whereas m should be in that context. If $V(n) = \{\{A\ B\}\ \{C\}\}$ for example, then if {A C} or {B C} are not in E then m should be in the context of E.

To capture this intuition the set of assumptions $\mathcal{A}$ that are defined for a particular ATMS-IDN are augmented ($\mathcal{A}^+$) to include *negated assumptions* of the form $\neg A$ for $A \in \mathcal{A}$:

$$\mathcal{A}^+ = \mathcal{A} \cup \{ \neg A \mid A \in \mathcal{A} \}.$$

As with a normal ATMS the values assigned to nodes are sets of sets of assumptions: $\mathcal{V} = Pw(Pw(\mathcal{A}^+))$. Each environment can be split into normal and negated assumptions, $E = E^+ \cup E^-$. Given a particular environment E, a node is in E's context iff for some $E_i \in V(n)$, $E_i^+ \subseteq E$ and $\neg E \cap E_i^- = \varnothing$ where $\neg E = \{\neg A \mid A \in E\}$.

As with the standard ATMS each node should be labelled with the minimal consistent environments capable of deriving that node. As before, E_2 is subsumed by E_1 iff $E_1 \subseteq E_2$. I.e. E_1 is more general than E_2 if it places fewer restrictions on the set of assumptions needed to derive it, be they positive (e.g. A must be in the current context) or negative (e.g. $\neg A$ must be in which means that A must not be in). Any label environment that is subsumed by another environment is removed from a node's label as are any environments that are subsumed by a no-good. Additionally, if an environment contains an assumption and its negation, where $\neg(\neg A) = A$, then that environment is also removed.

Given a standard support list justification $V_{in} W_{out} \longrightarrow_{SL} z$ the labels of the out-nodes B are processed at the antecedent stage to generate a set of negated environments to be combined at the dependency level with the unchanged in-node labels.

$S^{\mathbf{A}_{in}}(v_1, ..., v_p) = \{negate(v_1), ..., negate(v_p)\} = \{ \neg E \mid E \in combine(\{E_1, ..., E_q\}) \}$

where the v_i's are the values assigned to the nodes in V,

$negate(\{E_1, ..., E_q\}) = \{ \{ \neg x_1, ..., \neg x_q \} \mid (x_1, ..., x_q) \in E_1 \times ... \times E_q \}$

and $\neg\{a, ..., z\} = \{\neg a, ..., \neg z\}$

$S^{\mathbf{A}_{out}}(w_1, ..., w_p) = \bigcup_{1,p} \{w_i\}$ where the w_i's are the values assigned to the nodes in V

$S^{\mathbf{D}}(v_1, v_2) = remove(reduce(combine(v_1 \cup v_2)), \text{no-goods})$

where *remove* and *reduce* are as previously defined in §3.3.3 and *combine* is defined as follows:

$combine(\{L_1, ..., L_r\}) = \{ \{E_1, ..., E_r\} \mid (E_1, ..., E_r) \in L_1 \times ... \times L_r \}$

When describing the normal ATMS as an IDN the no-goods were passed to each dependency via a non-monotonic link, $\perp_{out}$ which were then explicitly passed to $S^{\mathbf{D}}$. In this case this link could be retained but in order to reduce the number of negated assumptions that would otherwise be created, the no-goods are implicitly passed to each dependency as shown in the definition of $S^{\mathbf{D}}$.

5.1.2 Example

Consider a problem solver that gathers evidence for three disjoint propositions π, ψ and ρ where π = "the train is on time", ψ = "the train is approximately five minutes late" and ρ = "the train is more than five minutes late". These propositions in turn provide support for π' = "the train will arrive on time", ψ' = "the train will arrive five minutes late" and ρ' = "the train will be more than five minutes late". Additionally the problem solver wishes to order the possible arrival times so that π' is preferred over ψ' which in turn is preferred over ρ'. I.e. if there is no other support for any of π, ψ or ρ then π is believed, and if there is support for more than one of π, ψ or ρ then π will be believed instead of ψ and ρ and ψ will be believed in preference to ρ.

This could be encoded by a set of dependencies $\pi_{in} \longrightarrow_{SL} \pi'$, $\psi_{in}\pi_{out} \longrightarrow_{SL} \psi'$, $\rho_{in}\{\pi\ \psi\}_{out} \longrightarrow_{SL} \rho'$, $\{\pi\ \psi\ \rho\}_{out} \longrightarrow_{SL} \pi'$.

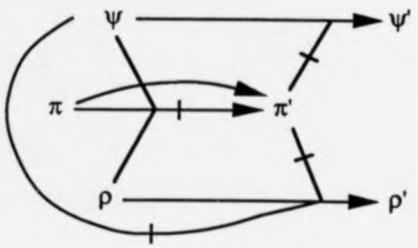


Figure 5.1.2 (a)

These dependencies could be interpreted as JTMS dependencies in which case for any admissible valuation V (assuming dependencies might be added to support π, ψ or ρ) the following (partial) valuations would be admissible.

π	*in*	*out*	*out*	*out*
ψ	*in/out*	*in*	*out*	*out*
ρ	*in/out*	*in/out*	*in*	*out*
π′	*in*	*out*	*out*	*in*
ψ′	*out*	*in*	*out*	*out*
ρ′	*out*	*out*	*in*	*out*

If NATMS labels are used instead, then assuming the set of no-goods is empty and that {A ... E} is some set of arbitrary assumptions, either of the following partial valuations might be admissible:

	V_1	V_2
π	{{A}}	{{A}{E}}
ψ	{{B}}	{{B}}
ρ	{{C}}	{{C D}}
π′	{{A}{¬A ¬B ¬C}}	{{A}{E}{¬A ¬E ¬B ¬C} {¬A ¬E ¬B ¬D}}
ψ′	{{B ¬A}}	{{B ¬A ¬E}}
ρ′	{{C ¬B ¬A}}	{{C D ¬A ¬B ¬E}}

Using resolution internally on the labels (another way to minimise the size and number of environments) these can be further reduced:

	V'_1	V'_2
π	{{A}}	{{A}{E}
ψ	{{B}}	{{B}}
ρ	{{C}}	{{C D}}
π'	{{A}{¬B ¬C}}	{{A}{E}{¬B ¬C}{¬B ¬D}}
ψ'	{{B ¬A}}	{{B ¬A ¬E}}
ρ'	{{C ¬B ¬A}}	{{C D ¬A ¬B ¬E}}

Compare this with the following extended ATMS (EATMS) encoding of the non-monotonic SL-dependencies which produces a set of monotonic SL-dependencies. Note - as all the dependencies are monotonic and all nodes are therefore in-nodes, the subscripts $_{in}$ and $_{SL}$ have been omitted for clarity! In this encoding the node and the corresponding (out-) assumption O_Γ is used to represent the (in-) assumption that all the nodes in the set Γ are *out*, while I_Γ represents the assumption that at least one of the nodes is *in*.

$\{\pi\ \psi\ \rho\}_{out} \longrightarrow_{SL} \pi' \equiv$

$O_{\pi\psi\rho} \longrightarrow \pi'$, $I_{\pi\psi\rho} \longrightarrow i_{\pi\psi\rho}$, $O_{\pi\psi\rho}\, i_{\pi\psi\rho} \longrightarrow \bot$, $\pi \longrightarrow i_{\pi\psi\rho}$, $\psi \longrightarrow i_{\pi\psi\rho}$, $\rho \longrightarrow i_{\pi\psi\rho}$

$\rho_{in}\{\pi\ \psi\}_{out} \longrightarrow_{SL} \rho' \equiv$

$\rho\ O_{\pi\psi} \longrightarrow \rho'$, $I_{\pi\psi} \longrightarrow i_{\pi\psi}$, $O_{\pi\psi}\, i_{\pi\psi} \longrightarrow \bot$, $\pi \longrightarrow i_{\pi\psi}$, $\psi \longrightarrow i_{\pi\psi}$

$\psi_{in}\pi_{out} \longrightarrow_{SL} \psi' \equiv$

$\psi\ O_\pi \longrightarrow \psi'$, $I_\pi \longrightarrow i_\pi$, $O_\pi i_\pi \longrightarrow \bot$, $\pi \longrightarrow i_\pi$

Additionally, in order to get the in- and out-assumptions for each set of out-nodes (i.e. I_Γ and O_Γ for $\Gamma \in \{\{\pi\ \psi\ \rho\}\{\pi\ \psi\}\{\pi\}\}$) to interact properly it is necessary to add additional meta-level operators *choose*($\{I_\Gamma O_\Gamma\}$) and *ignore*(I_Γ) for each Γ resulting in three *choose* constraints and three *ignore* constraints. This generates the following admissible valuation corresponding to V'_1 for the NATMS:

π $\{\{A\}\}$

ψ $\{\{B\}\}$

ρ $\{\{C\}\}$

I_Γ $\{\{I_\Gamma\}\}$

O_Γ $\{\{O_\Gamma\}\}$

$i_{\pi\psi\rho}$ $\{\{I_{\pi\psi\rho}\}\{A\}\{B\}\{C\}\}$

$i_{\pi\psi}$ $\{\{I_{\pi\psi}\}\{A\}\{B\}\}$

i_π $\{\{I_\pi\}\{A\}\}$

π' $\{\{A\}\{O_{\pi\psi\rho}\}\}$

ψ' $\{\{B\ O_\pi\}\}$

ρ' $\{\{C\ O_{\pi\psi}\}\}$

$\bot$ $\{\{O_{\pi\psi\rho}I_{\pi\psi\rho}\}\{O_{\pi\psi\rho}A\}\{O_{\pi\psi\rho}B\}\{O_{\pi\psi\rho}C\}\{O_{\pi\psi}I_{\pi\psi}\}$
$\{O_{\pi\psi}A\}\{O_{\pi\psi}B\}\{O_\pi I_\pi\}\{O_\pi A\}\}$

The number of no-goods generated is quite large and this is of particular interest given the desire to avoid a large set of no-goods as this increases the computational complexity [Dressler 1990].

Rather than having six nodes with four dependencies and an average environment size of 12/7 and an average label size of 7/6 which results from an NATMS, the standard EATMS results in 16 nodes, 16 dependencies, an average environment size of 7/5 and an average label size of 2. The average environment size and the average label size (excluding $\bot$) have decreased but the network is more complex and there are more no-goods.

For the second NATMS valuation V'_2, the EATMS gives the following admissible valuation:

π $\{\{A\}\{E\}\}$

ψ $\{\{B\}\}$

ρ $\{\{C\ D\}\}$

I_Γ $\{\{I_\Gamma\}\}$

O_Γ $\{\{O_\Gamma\}\}$

$i_{\pi\psi\rho}$ $\{\{I_{\pi\psi\rho}\}\{A\}\{E\}\{B\}\{C\ D\}\}$

$i_{\pi\psi}$ $\{\{I_{\pi\psi}\}\{A\}\{E\}\{B\}\}$

i_{π} $\{\{I_{\pi}\}\{A\}\{E\}\}$

π' $\{\{A\}\{E\}\{O_{\pi\psi\rho}\}\}$

ψ' $\{\{B\ O_{\pi}\}\}$

ρ' $\{\{C\ D\ O_{\pi\psi}\}\}$

$\perp$ $\{\{O_{\pi\psi\rho}\ I_{\pi\psi\rho}\}\ \{O_{\pi\psi\rho}\ A\}\ \{O_{\pi\psi\rho}\ E\}\ \{O_{\pi\psi\rho}\ B\}\ \{O_{\pi\psi\rho}\ CD\}\ \{O_{\pi\psi}\ I_{\pi\psi}\}$
$\{O_{\pi\psi}\ A\}\ \{O_{\pi\psi}\ E\}\ \{O_{\pi\psi}\ B\}\ \{O_{\pi}\ I_{\pi}\}\ \{O_{\pi}\ A\}\ \{O_{\pi}\ E\}\}$

For the EATMS the average size of environment remains approximately 4/3 with a label size, on average, 5/2. This compares with 7/3 and 10/6 for the non-resolution-reduced and 5/3 and 2 for the resolution-reduced NATMS. The size and number of EATMS environments will rise less quickly than for the NATMS with the trade-off being the increase in the number of no-goods. Conversely the NATMS does not produce any no-goods but has bigger increases in the size and number of environments, given an increase in the size and number of environments in the out-nodes' labels.

As an aside, it should be noted that it is possible to order three assumptions A, B and C at the meta-level quite easily with three *choose*s and two *ignore*s and five no-goods:

choose($\{A\ I_{BC}\}$) *choose*($\{I_{BC}\ O_{BC}\}$) *choose*($\{O_{BC}\ B\ C\}$) *ignore*(I_{BC}) *ignore*(C)
no–good($\{I_{BC}\ O_{BC}\}$) *no–good*($\{I_{BC}\ A\}$) *no–good*($\{O_{BC}\ B\}$) *no–good*($\{O_{BC}\ C\}$) *no–good*($\{B\ C\}$).

A is preferred to I_{BC} and only if A is no-good or inconsistent is I_{BC} chosen in preference to O_{BC} resulting in B being chosen ahead of C, unless of course B is itself contradictory. However, this encoding works at the meta-level, i.e. at the level of constraints on the assumptions and not on the nodes themselves. It is impossible in the EATMS to place a preference ordering on three **nodes** in the object level network purely through using *choose*, *ignore* and *no–good* constraints.

5.1.3 Interpretation Construction

The title of this section is somewhat of a misnomer: given an admissible valuation that assigns each node a label, one task that the problem solver may request is the construction of the context corresponding to a given problem solving environment E. In the basic ATMS this

simply requires a subsumption test for each node to see if one of the environments in its label is subsumed by E. In the EATMS it is necessary to construct a (potentially) expanded environment E′ that satisfies all the necessary meta-level constraints before carrying out the subsumption tests.

Using the first valuation from §5.1.2, if E = {C} then the constraints will propose the preferred extension E′ = {C $O_{\pi\psi\rho}$ $O_{\pi\psi}$ O_{π}}. However, given {$O_{\pi\psi\rho}$ C} is a no-good it is necessary to substitute $I_{\pi\psi\rho}$ for $O_{\pi\psi\rho}$. This gives the extended environment E′ = {C $I_{\pi\psi\rho}$ $O_{\pi\psi}$ O_{π}}, from which the following context results: {ρ ρ′ O_{π} $O_{\pi\psi}$ $I_{\pi\psi\rho}$ $i_{\pi\psi\rho}$}.

This compares with the NATMS where the environment E = {C} generates the context {ρ ρ′} through the standard subsumption test.

Using E = {A C} the meta-level constraint satisfaction process for an EATMS is even more involved. Each preferred choice of $O_{\pi\psi\rho}$ over $I_{\pi\psi\rho}$, $O_{\pi\psi}$ over $I_{\pi\psi}$ and O_{π} over I_{π} leads to a contradiction as each of {$O_{\pi\psi\rho}$ A}, {$O_{\pi\psi}$ A} and {O_{π} A} are no-goods. This leads to E′ = {A C $I_{\pi\psi\rho}$ $I_{\pi\psi}$ I_{π}} and a context {π π′ ρ I_{π} i_{π} $I_{\pi\psi}$ $i_{\pi\psi}$ $I_{\pi\psi\rho}$ $i_{\pi\psi\rho}$}.

5.1.4 Conclusion

Having outlined the NATMS it is possible to see the benefits outlined in §5.1. The NATMS is more expressive than a standard ATMS in allowing non-monotonic justifications. This in turn allows for more complex interactions between nodes to be captured. Despite the ATMS's emphasis on making choices it is not possible to define orders on the choice of assumptions to be used.

By introducing meta-level constraints (*choose* and *ignore*) on how contexts should be constructed from environments it becomes possible in an EATMS to introduce non-monotonic justifications. It is also possible to order choices on assumptions purely through using these constraints, although it is not possible to directly define choices between ordinary nodes in this way. However, as seen above, the coding scheme for non-monotonic justifications is cumbersome, needing a large number of extra nodes, dependencies and assumptions. This approach is also less elegant for it is necessary to introduce a secondary constraint satisfaction phase between specifying a problem solving environment and

constructing its context.

The NATMS avoids these problems: non-monotonic SL-justifications can be directly represented in the network, there is no need for extra network or meta-level constructs to be added; interpretation of the network is done at the network level through standard propagation algorithms and subsumption tests; and there is no need for a meta-level constraint satisfaction process.

On the efficiency side, it is hard to judge whether an IDN-based NATMS is more efficient than the EATMS encoding of non-monotonic dependencies. As shown in §5.1.2 there is a trade-off between size of labels and the number of no-goods and without further detailed experiments on networks of varying connectedness, there is no straight forward way to see which, if either, would be the dominant factor.

Finally, by directly representing SL-dependencies in the network it is possible to provide J- and ATMS-style Truth Maintenance over the same network by changing the interpretation structure $<\mathcal{S}, \mathcal{V}>$. In this way we can see different styles of truth maintenance as providing different semantics for the same network. The JTMS interpretation is of standard propositional form with simple assignments of truth or falsity while the ATMS is in a many-worlds style with assignments giving an indication as to the worlds in which a given proposition is true.

5.1.5 Multiple Interpretations (again)

One obvious problem that is encountered with extended or non-monotonic ATMSs is that of non-monotonic cycles. In a JTMS even cycles give multiple admissible valuations while odd cycles can result in none. Given there is a (partial) mapping between JTMS and ATMS interpretations[1] it should be no surprise that it is possible to get no or multiple admissible valuations with the NATMS. Given an odd loop $a_{out} \longrightarrow_{SL} a$ it is only possible to generate an admissible valuation if there is some other dependency d attached to a and $V^{\mathbf{D}}(d, V) = \{\{\}\}$. Given an even loop $a_{out} \longrightarrow_{SL} b$ and $b_{out} \longrightarrow_{SL} a$ multiple interpretations arise. If there is no

[1] V is an admissible JTMS valuation iff V′ is an admissible ATMS valuation where $V(n) = in \leftrightarrow V'(n) = \{\{\}\}$ and $V(n) = out \leftrightarrow V'(n) = \{\}$.

support for either a or b in a particular context then in one admissible valuation a will be in the context while in the other valuation b would be in the context.

Example

Consider the following NATMS network with its possible admissible valuations:

$D = \{a_{out} \longrightarrow_{SL} b,\ b_{out} \longrightarrow_{SL} a,\ \longrightarrow_{\{\{A\}\}} a,\ \longrightarrow_{\{\{B\}\}} b\}$

$V_1(a) = \{\{A\}\{\neg B\}\},\ V_1(b) = \{\{B\}\}$

$V_2(a) = \{\{A\}\},V_2(a) = \{\{\neg A\}\{B\}\}.$

Environment	**Context$_1$** (using V_1)	**Context$_2$** (using V_2)
{}	{a}	{b}
{A}	{a}	{a}
{B}	{b}	{b}
{A B}	{a b}	{a b}

This is one area where it might appear that EATMS scores over the NATMS. The corresponding EATMS expansion of the non-monotonic dependencies gives the following acyclic network with just a single admissible valuation.

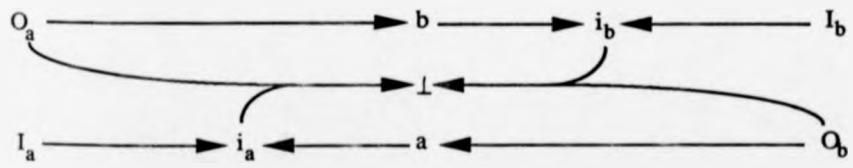


Figure 5.1.5 (a)

choose($\{O_a\ I_a\}$)

choose($\{O_b\ I_b\}$)

ignore($\{I_a\}$)

ignore($\{I_b\}$)

$V(O_a) = \{\{O_a\}\}$

$V(I_a) = \{\{I_a\}\}$

$V(i_a) = \{\{I_a\}\{O_b\}\}$

$V(O_b) = \{\{O_b\}\}$

$V(I_b) = \{\{I_b\}\}$

$V(i_b) = \{\{I_b\}\{O_b\}\}$

$V(a) = \{\{O_b\}\}$

$V(b) = \{\{O_a\}\}$

$V(\bot) = \{\{O_a\ I_b\}\ \{O_b\ I_a\}\ \{O_b\ O_a\}\}$

Given any problem solving environment E the *choose/ignore* constraints will attempt to force the inclusion of one or both of O_a and O_b in the extended environment E´. As the inclusion of both is contradictory, two extensions result: $E'_1 = \{O_a\ I_b\}$ and $E'_2 = \{O_b\ I_a\}$. Depending on which one of E'_1 or E'_2 is picked, a context is generated that contains b or a respectively. So although the problem of multiple admissible valuations is avoided, the problem of multiple contexts remains. In this situation either the user has direct control over which extension is picked or an arbitrary choice must be made by the system[2].

Another problem that the EATMS solves, enabling the construction of an admissible valuation, is that of non-monotonic cycles. The expansion of non-monotonic dependencies in an odd loop will generate a stabilising assumption, allowing the cycle to be interpreted. In other words, given an odd loop $a_{out} \longrightarrow_{SL} a$ the EATMS will assume there is independent support for a. This means a will be included in any context, and the odd cycle does not occur. The expansion of $a_{out} \longrightarrow_{SL} a$ and its admissible valuation (see below) demonstrate this.

[2] This arbitrary choice is exactly one of the reasons cited against the JTMS. However this problem will invariably arise when a system includes non-monotonic justifications. If non-monotonic cycles are allowed, multiple interpretations will follow unless an arbitrary (system) choice is made.

Example

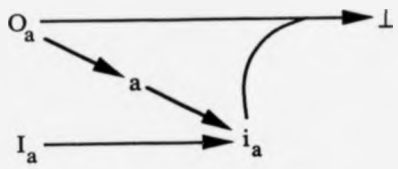


Figure 5.1.5 (b)

choose($\{O_a\ I_a\}$)

ignore($\{I_a\}$)

$V(O_a) = \{\{O_a\}\}$

$V(a) = \{\{O_a\}\}$

$V(I_a) = \{\{I_a\}\}$

$V(i_a) = \{\{I_a\}\{O_a\}\}$

$V(\perp) = \{\{O_a\}\}$

Given O_a is contradictory, any problem solver environment is forced to include I_a, thus a will be in all contexts. Here the EATMS is forcing "a" to be *in* to avoid what would otherwise be a "contradiction" or unsatisfiable "cycle" (there is no actual cycle in the network itself). Because the translation scheme actually removes the dependency between nodes in any non-monotonic cycle it is possible to generate a single admissible valuation of the network, even in the case of even non-monotonic cycles where one might expect multiple possible interpretations.

So it can be seen that at least some of the problems of the NATMS are shared with existing EATMSs. One problem that is unique to the NATMS is the number and size of environments created by non-monotonic antecedents. Some further work should be done on investigating the possibility of abbreviating environments so that an environment $\{A_1\ A_2 A_3\}$ might be replaced by $\{A_{123}\}$, so a non-monotonic link would generate an environment $\{\neg A_{123}\}$ rather than the environments $\{\neg A_1\}$, $\{\neg A_2\}$ and $\{\neg A_3\}$.

5.2 Default Reasoning

There is a very close connection between default reasoning and Truth Maintenance Systems. Indeed, the JTMS [Doyle 1979] can be thought of as an implementation mechanism for NML-I [McDermott and Doyle 1980]. Additionally, as mentioned in §1.2.2, there are strong associations between NML-I and autoepistemic logic (AE) [Moore 1983] and it has been demonstrated [Konolige 1986] that autoepistemic logic and Reiter's [1980] default logic (DL) are of equivalent power and expressiveness.

In general a default rule (in DL, see 1.2.2) of the form α: Mβ / γ is equivalent to an autoepistemic implication L$\alpha \wedge \neg$L$\neg\beta \rightarrow \gamma$ which in turn is equivalent to $\neg$M$\neg\alpha \wedge$ M$\beta \rightarrow \gamma$ (taking M in NML as the dual of L whereby M $\equiv \neg$L$\neg$). These translations are somewhat counter-intuitive. One would expect an intuitive reading of the Reiter default to be "if α is true and it is consistent to assume β then γ is true". Yet the intuitive reading of the corresponding NML-I implication is "if $\neg\alpha$ is inconsistent and it is consistent to assume β then γ is true". The truth condition on α has become somewhat distorted in form even if it is equivalent in content.

The intuitive reading of the autoepistemic implication ("if α is believed (known) and $\neg\beta$ is not believed (known) then γ is true") is at first glance even more counter-intuitive: why should α not have to be true but be believed to be true? The answer comes from the semantics of the default. Take the standard flying bird and the default that says normally (or normal) birds fly. This can be interpreted as saying only those birds explicitly stated not to fly don't actually fly. To be able to say a bird flies because it is not explicitly stated not to fly relies on the fact that not only is the object in question a bird, it is actually *known* or believed to be a bird. However this point becomes secondary when the defaults considered are of the form Mβ / γ.

Morris [1987] explicitly links defaults and Truth Maintenance in proposing the use of the JTMS as a default reasoning device. This is a departure from the normal problem solver/JTMS interaction whereby the TMS does actually instantiate "default" rules but only in the creation of assumptions to guide the search process and control dependency directed backtracking. By coding defaults as dependencies Morris claims that the problem of

anomalous extensions can be cured. Consider the following problem (a non-temporal version of the Yale Shooting Stick problem devised by Morris): normal animals (nA) can't fly, winged animals (W) are abnormal animals (abA) (where abnormal ≡ not normal) with respect to the property of not flying (i.e. winged animals can normally fly which does not fit the animal norm of not flying), all birds (B) are animals and normal birds (nB) have wings. Does Tweety the bird fly? Translating this into (Reiter's) default logic produces the following implications[3]:

$A \wedge nA \rightarrow \neg F$

$W \rightarrow abA$

$B \rightarrow A$

$B \wedge nB \rightarrow W$

The default rules or assumptions capture the intuition that things are "normally" normal! I.e. if it is consistent to assume normality (there is no evidence of abnormality) then assume normality:

$:MnA / nA$ (RA) $:MnB / nB$ (RB)

In default logic (be it Reiter's as illustrated above, or Moore's or McDermott and Doyle's) adding the fact that Tweety is a bird produces two extensions. In the first RB (or the equivalent) is used to assume the normality of Tweety as a bird which in turn implies Tweety has wings and is therefore an abnormal animal w.r.t. flying and the system is agnostic as to whether Tweety flies or not - as Tweety is an abnormal animal RA cannot be applied. In the second RA is used to assume normality of Tweety as an animal implying that Tweety cannot fly. As a side effect, Tweety's normality implies (contra-positively) that Tweety doesn't have wings and is therefore an abnormal bird with respect to having (or not having) wings.

The default theory can be converted into dependencies (in the spirit of Morris) by converting each implication into a monotonic dependency with each proposition represented

[3] This axiomatisation omits some obvious implications, e.g. $W \rightarrow F$. However, this does not affect Morris' example which centres around which defaults take precedence, and whether or not multiple interpretations are possible.

by a node[4]; the default rules are represented as non-monotonic justifications so that if abA is *out* (abA ↔ ¬nA) then n is *in*.

$\{A , nA\}_{in} \longrightarrow_{SL} \neg F$

$\{W\}_{in} \longrightarrow_{SL} abA$

$\{B\}_{in} \longrightarrow_{SL} A$

$\{B , nB\}_{in} \longrightarrow_{SL} W$

$\{abA\}_{out} \longrightarrow_{SL} nA$

$\{abB\}_{out} \longrightarrow_{SL} nB$

This network, being acyclic, gives rise to a single admissible valuation and lo and behold the intuitive answer (that Tweety the bird does not necessarily not fly) is supported in this valuation.

nodes	A	W	B	¬F	abA	abB	nA
V	*in*	*in*	*in*	*out*	*in*	*out*	*out*

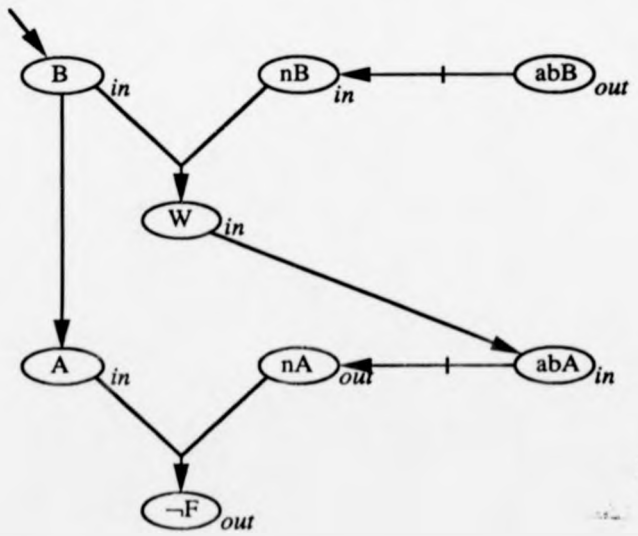


Figure 5.2 (a)

[4] The scheme represented here is a propositional case, but as shown in §5.2.4 there are a variety of techniques for converting propositional representations into predicate representations.

This example works for the following reason: the semantics of the TMS puts an implicit ordering on the abnormality assumptions and the one way nature of support makes it impossible for the justifications to be used in a contra-positive way. A more realistic set of dependencies to model the set of assumptions would include the contra-positives of the implications, in particular:

$$\{B\ ,\ \neg W\}_{in} \longrightarrow_{SL} abB$$

$$\{nA\}_{in} \longrightarrow_{SL} \neg W$$

This is not motivated by the desire to follow the default logic reasoning but by intuitive notions of "normal" birds and animals and their lack, or otherwise, of wings.

The addition of these two dependencies creates a cycle (a necessary condition for multiple interpretations in deterministic IDNs - see §4.1.4) and two admissible valuations exist for this network.

nodes	A	W	B	¬F	aA	aB	nA
V_1	*in*	*in*	*in*	*out*	*in*	*out*	*out*
V_2	*in*	*out*	*in*	*in*	*out*	*in*	*in*

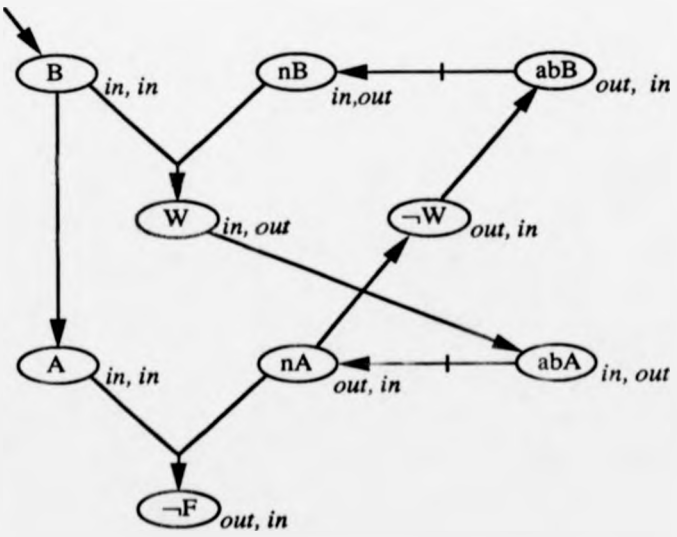


Figure 5.2 (b)

The problem of unwanted extra interpretations has reappeared.

Given the position of this thesis, i.e. that dependencies are not just records of inferences but can actually be used to **make** inferences, it is hardly surprising that by being selective about the set of inferences used to capture certain knowledge one can achieve the desired results.

In standard first order or propositional logic the inferences that can be made from a given set of propositions are given by the inference rules which in turn (certainly in the case of natural deduction systems [Lemmon 1965]) are derived from the (intuitive/truth functional) semantics of the connectives. Proof theoretic default or non-monotonic logics are also trying to capture the intuitive notion of defaults by adding logical machinery that allows or generates the intuitive inferences from standard constructs. One cannot be arbitrary in the choice of logical construct, changing the forms on an ad hoc basis to achieve the desired results.

In the following section I shall present a systematic representation of defaults that captures a particular set of intuitions about defaults that gives plausible results (in terms of the inferences made - even when considering sets of defaults) and an easy way of specifying interactions between defaults.

5.2.1 The Structure of Defaults

Consider some of the forms used in the literature on default or non-monotonic logics: birds fly by default; normally birds fly; normal (prototypical) birds fly; typically birds fly. There are different possible interpretations of very similar forms - is the default expressing the fact that the consequent holds for the majority of objects satisfying the antecedent, or for the normative examples satisfying the antecedent, or for some specified set of "normal" individuals? I take the approach that the latter should be the normal reading of a default - any bird that is believed to be normal (with respect to flying) will fly.

This "normality" check is not a simple (!) appeal to the ability to consistently assert the desired conclusion - which in this case would correspond to the fact "Tweety flying" being consistent with him being a bird and an animal and anything that can be derived from these

facts. The default is a strict (material) implication and any bird that doesn't fly cannot simultaneously be normal. The default implication comes in assuming that unless there is evidence of abnormality then normality can be assumed. If a bird is known not to fly then it is obviously abnormal w.r.t flying and the normality assumption cannot be made and therefore the inference that it can fly does not go through.

This set of inferences corresponds to the following set of justifications[5] for a default rule of the form "A_1 ,.... A_n normally implies B":

$$\{A_1 \text{ ,....}, A_n, n_B A\}_{in} \longrightarrow_{SL} B$$

$$a_B A_{out} \longrightarrow_{SL} n_B A$$

$$\{A_1 \text{ ,....}, A_n, \neg B\}_{in} \longrightarrow_{SL} \neg n_B A$$

If one considers the TMS as storing beliefs (that are held by some abstract or actual agent) about the world then the antecedents have the same strength as those in an Auto-Epistemic (AE) default rule - they only have to be believed to be true (instead of being true - a $\rightarrow$ La but not necessarily vice versa). The consequent however is only a belief about the world not an actual statement of fact and in this sense is weaker than the AE consequent. This however accords with most people's intuitions about defaults: given agents in the world are not omniscient nor ideally introspective, people may have false beliefs and the conclusions (of default rules) are not statements of fact. If I believe that Tweety is a normal bird then I believe that Tweety can fly. To actually know that Tweety can fly requires that I actually know Tweety is a normal bird, which in turn requires absolute certainty in the soundness of my belief that Tweety is not a normal bird and that if he were abnormal then I **would** know about it.

The contra-positive form $\{A_1 \text{ ,....}, A_n, \neg B\}_{in} \longrightarrow_{SL} \neg n_B A$ is not motivated by a desire to reintroduce multiple extensions in Morris' example (although it does do this) but because it is natural to allow contra-positive reasoning. This particular form was chosen because it is valid and the other forms (e.g. $\{A_2 \text{ ,....}, A_n, b_B A, \neg B\}_{in} \longrightarrow_{SL} \neg A_1$) are counter-intuitive. To

[5] This representation is an extension or development from the use of abnormality predicates that first appeared in connection with circumscription [McCarthy 1980].

say that if something doesn't fly but is a normal bird (with respect to flying) then it can't actually be a bird seems slightly nonsensical - being a normal (or abnormal) bird in the first place pre-supposes that the individual in question actually **is** a bird to start with[6].

5.2.2 Interacting Defaults

The case of Tweety the winged not non-flying animal is one particular example of interacting defaults (see for example [Reiter and Criscuolo 1987]). When more than one default is present in a given theory they may interact in counter-intuitive ways or not interact at all. How defaults interact has less to do with the way defaults are linked but more with intuitive notions of what inferences should follow.

Consider the Nixon diamond: normal Quakers (nQ) are doves (D), normal Republicans (nR) are hawks (H - the opposite or negation of being a dove), is Dick (the Republican Quaker) a dove or a hawk (or neither or both)? Leaving aside the fact that any bird is also an animal, the Nixon diamond has the same form as Tweety the bird: normal birds fly, normal animals don't fly, does Tweety the bird (who is also an animal) fly? The intuitive responses to the Nixon diamond would be to either remain agnostic about whether Dick were a hawk or say that either result could hold (though not simultaneously), or arbitrarily pick either one of the two. This is not the case when considering whether Tweety can fly where the intuitive answer is that he can of course fly.

---

[6] Giordano and Martelli [1990] take the opposite tack and use the negation of the consequent to infer that a normal antecedent (i.e. an antecedent other than the normality node or assumption in the JTMS sense of the word) must be false. What is interesting about their approach is that their three-valued interpretations and two way use of dependencies results in a system very much like the LTMS [McAllester 1978]. It would be interesting to investigate whether the addition of a (weak) non-monotonic negation operator (rather than the strict negation that is used) would result in semantically and functionally isomorphic systems.

$\{A, nA\}_{in} \longrightarrow_{SL} \neg F$ $\quad \{abA\}_{out} \longrightarrow_{SL} nA$ $\quad \longrightarrow_{SL} A$

$\{B, nB\}_{in} \longrightarrow_{SL} F$ $\quad \{abB\}_{out} \longrightarrow_{SL} nB$ $\quad \longrightarrow_{SL} B$

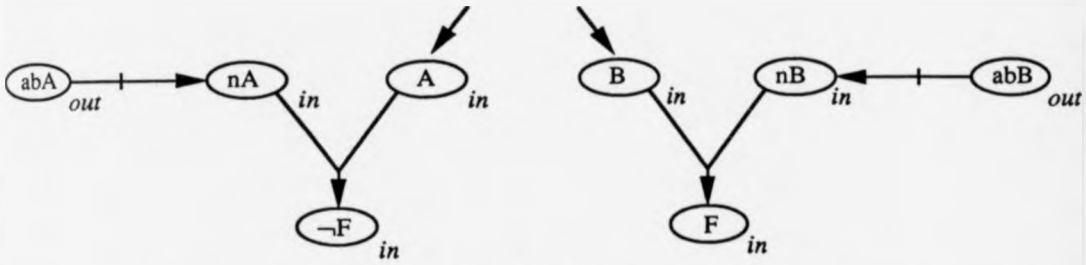


Figure 5.2.2 (a)

$\{R, nR\}_{in} \longrightarrow_{SL} H$ $\quad \{abR\}_{out} \longrightarrow_{SL} nR$ $\quad \longrightarrow_{SL} R$

$\{Q, nQ\}_{in} \longrightarrow_{SL} D$ $\quad \{abQ\}_{out} \longrightarrow_{SL} nQ$ $\quad \longrightarrow_{SL} Q$

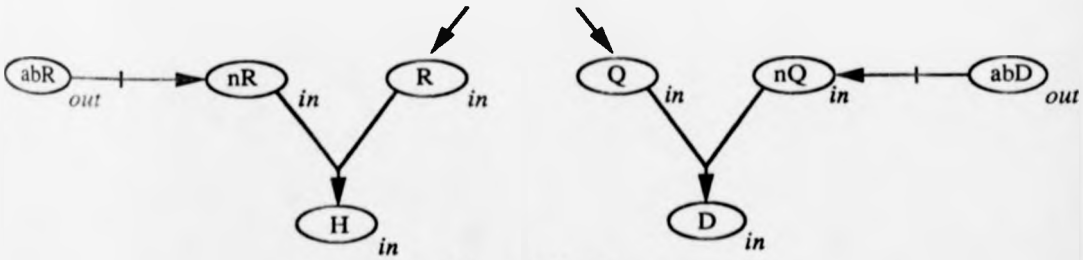


Figure 5.2.2 (b)

Given the same network above captures the basic information about the defaults in both examples, the interaction between the defaults cannot be a matter of basic form - the interactions themselves must be explicitly supplied and represented as extra dependencies. A system should not be expected to distinguish the different semantics of the two examples by magic!

Given the set of dependencies used to represent defaults, the interactions between defaults can be captured by dependencies attached to abnormality propositions (nodes). In the above examples the Nixon diamond doesn't warrant any more dependencies. This is in contrast to Tweety: birds are abnormal animals w.r.t not flying and the inclusion of this information by adding $\{B\}_{in} \longrightarrow_{SL} abA$ will remove the anomalous valuation[7].

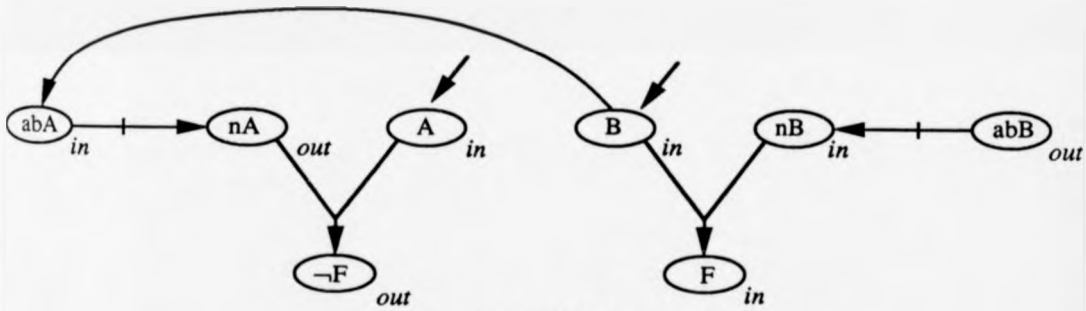


Figure 5.2.2 (c)

As an aside, it is interesting to note that if contra-positive forms were not used then a contradiction would be generated. If invoked, dependency directed backtracking (DDB) would find nA and nB were the assumptions (in the JTMS sense where an assumption is a node supported by a non-monotonic dependency) underlying this contradiction. The obvious assumption to revoke would be nA and this would be done by justifying aA. The explicit inclusion of $\{nB\}_{in} \longrightarrow_{SL} abA$ is capturing this process without explicitly needing to invoke DDB. The stronger antecedent (that birds, not just normal birds, are abnormal animals w.r.t

[7] This could be viewed as preferring the more specific default as proposed by Poole [198], but the whole point is that this doesn't **necessarily** have to be the case. As normality and abnormality are explicitly represented, many different ways of linking classes to default behaviours can be used.

flying) is used in order to construct a hierarchy of abnormalities. Penguins should not be normal animals w.r.t flying for this would imply that they don't fly through virtue of being an animal. Penguins don't fly because they are penguins and are as such abnormal birds w.r.t. flying, as well as being abnormal animals w.r.t. flying. In some sense, the contra-positive dependencies are negating the need for DDB - for DDB would add exactly this kind of dependency.

The initial presentation ignored the dependency between birds being animals. It might be claimed that the reason flying is preferred over non-flying is precisely because of this inference [e.g. Loui 1987]. The fact that birds are a subset of animals means that the normality of an animal in not flying should be over ridden by the normality of birds flying. In general this would suggest that whenever X implies Y and X is normally Z and Y is normally ¬Z, then a normal X should imply abnormality of Y, w.r.t. Z. This type of reasoning is hard to implement for it is necessary to have a contradiction between X and Y over Z before adding X implies abnormal (w.r.t. Z) Y[8]. If the inferences between X, Y and Z are not obvious and are the product of long chains of inferences then it is difficult to determine when a contradiction can be averted by adding the necessary dependencies linking types and super-types.

Another difficulty with interacting defaults is that the interactions between defaults cannot be uniformly generated by dependencies between the propositions in the defaults. A good example of this comes in considering the composition of two defaults: A normally implies B and B normally implies C. Without any additional information, asserting A will lead to the derivation of C. However, A may have some other explicit relationship with C. In particular it may be the case that A normally implies C (directly, without any information relating to B), or A doesn't normally imply C or an even stronger condition, A normally implies ¬C.[9]

[8] It should be noted that in Poole's scheme, the expensive work of determining which defaults get overridden only gets done when a contradiction **actually** occurs. It is not necessary to look for contradictions ahead of time. In this case, information about relationships between defaults is not actually stored as part of the knowledge being captured but is more a heuristic to be used in resolving contradictions as and when they arise

[9] The case of A not normally being ¬C is irrelevant to this case as there is nothing to suppose that A normally implies ¬C. In any case the representation of A not normally implying ¬C would be handled analogously to A not normally implying C.

If A normally implies C [Reiter and Criscuolo 1987, Eg 2.13 p274] then this default is coded in the normal fashion so that A's that are not B's (because the A's are abnormal w.r.t. B) are still C's: $\{A , n_C A\}_{in} \longrightarrow_{SL} C$, $\{A , \neg C\}_{in} \longrightarrow_{SL} ab_C A$, $\{ab_C A\}_{out} \longrightarrow_{SL} n_C A$.

If A's are not normally C's then transitivity must be blocked. In Default Logic this must be done by modifying the actual default rule to explicitly exclude A's that are B's from being C's [Reiter and Cruiscuolo 1987, Eg 2.14]:

$$\frac{B : MA}{C} \text{ becomes } \frac{B : M(\neg A \wedge C)}{C}$$

This has the following disadvantages:

a) defaults are not modular and the same default "B's are normally C's" can end up being represented by two different default rules, having been embedded in two different theories, to take account of different interacting defaults in the two different cases;

b) the addition of interacting defaults means changes to **existing** rules must be made and no default is guaranteed to be stable and retain its original semantics; and

c) as defaults are modified, in order to accommodate other conflicting defaults, they become ever more complex.

Using an IDN representation the transitivity is blocked by adding $A_{in} \longrightarrow ab_C B$ as A's cannot be normal B's w.r.t. C if A's are not C's. Any other default that can be used to derive C from A must also be blocked by asserting $A_{in} \longrightarrow ab_C X$ where X is the property that by default sanctions C.

If A's are normally $\neg$C's then a standard default representation should be added. This (normally) results in two possible admissible valuations as no priority is given to the default "A normally implies $\neg$C" over the defaults "A normally implies B" and "B normally implies C". To explicitly force a preference for $\neg$C over C the implicit inference that "A normally implies $\neg$C" entails "A doesn't normally imply C" has to be made and represented as above. This removes one of the admissible valuations leaving V(C) = *out* and V($\neg$C) = *in*.

5.2.3 Conclusion

One result of examining the link between Default Logic (DL) and TMSs is the conclusion that TMSs, simply by virtue of their functionality, do not solve the problem of multiple extensions. The JTMS (and IDNs in general) provide an expressive representation for capturing inferences, including default inferences, but this must be done in some principled way to accurately reflect intuitions. Morris [1987] does not do this in the case of his "solution" to the problem of anomalous extensions.

It is particularly important in the case of multiple defaults that intuitions about how they interact should be explicitly represented in a dependency network. There appears to be no way that these intuitions can be automatically generated, for example by simply considering orders on defaults or dependencies across defaults.

It should be noted that although the IDN representation scores in modularity and compositionality[10] over the DL representation given by Reiter and Criscuolo [1987], (where the default logic rules grow in size and complexity with each new interaction that occurs) in principle DL could be used to capture the IDN representation of defaults. E.g. if A normally implies B, the contra-positive

$$a \wedge n_B A \rightarrow B \quad \text{is equivalent to} \quad \frac{: M n_B A}{n_B A}$$

This observation highlights the main conclusions of this section: IDNs (in the form of the JTMS-IDN) are capable of representing default inferences[11] but this does not solve the multiple interpretation problem. What counts is the actual inferences represented. The representation of defaults in this section provides for a modular way of capturing relationships between defaults, and moves the default inference away from assuming that particular inferences are defaults e.g that birds fly by default, to assuming by default that things are normal.

[10] two of the major advantages cited for some AI representations e.g. in Woods' [1983] idea of "notational efficacy"

[11] It is not necessary to introduce novel three-valued interpretations and propagation schemes [Giordano and Martelli 1990] to achieve this.

What the IDN representation does do is highlight the natural occurence of cycles in default reasoning. These cycles are not only responsible for the occurence of multiple extensions both in the IDN representation (being a necessary but not sufficient condition) and in DL (through the choice of different application orders for default rules) but have an adverse affect on the computational complexity resulting in the cost of constructing admissible valuations becoming exponential in the worst case.

Finally, although Morris's solution to the multiple extension problem does not work he does highlight the need for a declarative semantics and temporally independent interpretation mechanism for the JTMS. This is yet another problem that the IDNs go some way towards solving.

5.2.4 Individuals and Universally Quantified Rules

In the preceding discussion, default inferences have been represented as dependencies between nodes where each node represents some proposition. There has been some subtle disingenuity with the translation of defaults to nodes and dependencies. "Birds normally fly" is translated into "normal birds fly" which is represented by a dependency $\{B , nB\}_{in} \longrightarrow_{SL} f$. Clearly "B" could be thought of as the property of being a bird and "nB" as being a normal bird so the dependency represents a default inference about properties. If this is the case then how are inferences about individual birds to be made?

Asserting Tweety is a bird should lead to the conclusion that Tweety can fly. Claiming "B" and "f" are the propositions "Tweety is a bird" and "Tweety can fly" leads to problems when considering Felix the finch. New nodes "B´", "nB´" and "f´" with dependency $\{B' , nB'\}_{in} \longrightarrow_{SL} f'$ would then be needed to make the default inference that Felix can fly. Really the default "birds normally fly" should be translated into a universally quantified sentence "$\forall x.\ B(x) \wedge nB(x) \rightarrow f(x)$" or an open formula "$B(x) \wedge nB(x) \rightarrow f(x)$" (as with open defaults [Reiter 1980]) which act as schema for particular inference rules given appropriate instantiations of variables.

How universally quantified sentences are to be dealt with is a question that arises for many IDN-based systems. For example, this topic comes up again in §5.3 when dealing with

Rule-Based Systems. This subject will be covered here purely for the sake of convenience and continuity with the representation and examples discussed in the rest of §5.2.

In general there appears to be two ways of approaching this problem - i.e. the process of instantiating universal formulae and reasoning with them. The first is to have an interface between IDN and user whereby queries or assertions capable of unifying with part of an open default are instantiated as instances of those rules. Thus assertions such as "B(Tweety)" or "B(Felix)" combined with the default rule[12] $\{B(x), nB(x)\}_{in} ==> f(x)$ produce dependencies $\{B(Tweety), nB(Tweety)\}_{in} \longrightarrow f(Tweety)$ and $\{B(Felix), nB(Felix)\}_{in} \longrightarrow f(Felix)$.

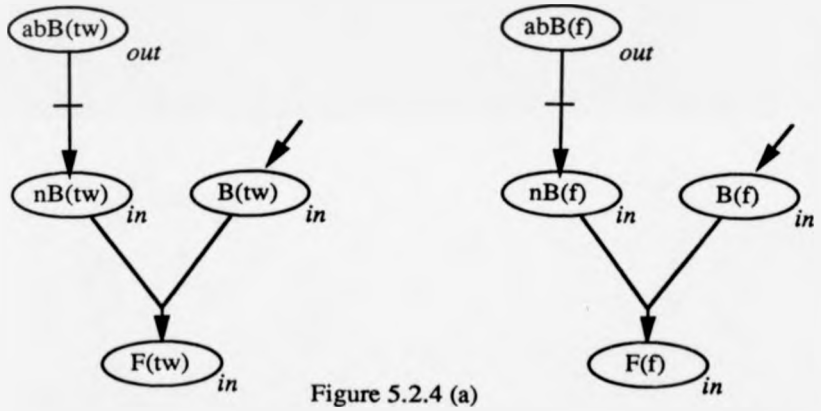


Figure 5.2.4 (a)

This approach requires a interface capable of storing, unifying and instantiating the dependency schemata. The schemata would not form part of the dependency network but would act as a mechanism for "packing" an expanded, potentially infinite, network consisting of all possible instantiated versions of a given schema. The drawback with this approach is that the schema is not represented in the network and without instantiating particular instances of dependencies it is impossible to look at the relationship between nodes. Therefore any user interaction must be capable of initiating instantiation. If an assertion is made that unifies with a dependency schema it must be instantiated as a query about the

[12] The double arrow ==> is used to represent dependency schemata as opposed to the single arrow $\longrightarrow$ which represents actual dependencies within a network.

status of a proposition or the relationship between two or more propositions. For example if it is known that all individuals are red unless otherwise indicated, $\{\text{not-red(x)}\}_{out} \Longrightarrow \text{red(x)}$, querying the status of "red(fox)" should instantiate the dependency thus supporting a value *in* for red(fox). One possible way of implementing this would be by using *consumers* (see Chapter 6).

The second (alternative) approach to representing quantified expressions is to represent the open default as a dependency between open formulae and incorporate the binding of variables in the interpretation mechanism. Rather than assign values from {*in*, *out*}, nodes are assigned a list of pairs: $\mathcal{V} = Pw(\{ (i, v) \mid i \in I, v \in \{in, out\} \})$ where I is a set of names of individuals. If Tweety and Felix are both birds but Tom (the cat) is not, the node B(x) should have the value V(B(x)) = {(tw, *in*) (b, *in*) (tm, *out*)}. The names in I act as an index for calculating admissible valuations for each individual: it is as if the dependency network were recreated for each individual without the need to actually perform the replication.

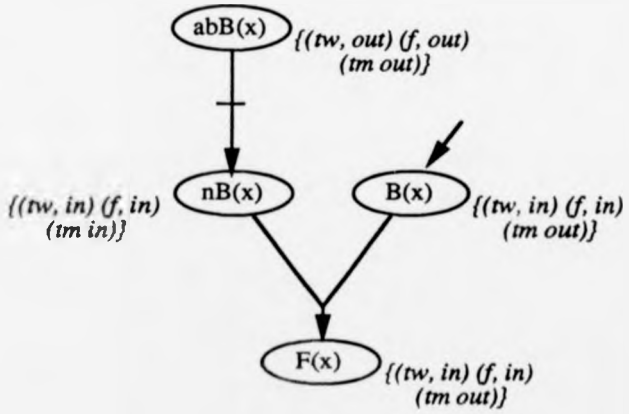


Figure 5.2.4 (b)

When a network is purely propositional or is taken to refer to a single individual, the assertional dependency $\longrightarrow_{SL} n$ is sufficient to assert that n is true. When dealing with networks with individuals it is necessary to include information about which individual some assertion applies to. This is done through assertional dependencies d of the form $\longrightarrow_{I} m$

where i ∈ *I*. The value of the d is simply an individual/value pair:

$$V^{\mathbf{D}}(d, V) = \{(i, in)\}.$$

The summation of the indexed values at all levels is done by grouping the values for each individual and then applying the standard JTMS summation function. For example, at the node level this is done by the summation function $S^{\mathbf{N}}$

$$S'^{\mathbf{N}}(v_1, \ldots, v_p) = \{(i, v) \mid i \in I, v = S^{\mathbf{N}}(v'_1, \ldots, v'_q) \text{ for } v' \in V_i\}$$
$$\text{where } V_i = \{ w \mid \exists j.(i, w) \in v_j \}$$

where $S^{\mathbf{N}}$ is the JTMS node summation function.

Using this approach each node will be labelled with a set of pairs such that each individual i ∈ *I* will be in exactly one pair. This approach works well if the predicates apply to a large number of the set of individuals, otherwise the equivalent of a large number of unnecessary nodes and dependencies would be added. Consider a set of inferences about animals, birds and their respective abilities to fly. Imagine the addition of rules that say that helicopters fly, are capable of carrying people and large cargoes, are used for air-sea rescue and anti-tank warfare, are noisy etc. One would not want to label all the nodes related by these dependencies as being *out* for all the individuals that are birds but not helicopters, purely on the grounds that birds, like helicopters, fly.

Applying the closed world assumption would mean that if (i, *in*) ∉ V(n) then (i, *out*) would have been *in* if the network were instantiated for i. The question is then how and when to apply the closed world assumption to generate non-monotonic inferences. This is done at the antecedent level so that individual antecedents not satisfying the dependency (i.e. an in-node is *out* or an out-node is *in*) are not even recorded as (i, *out*):

$$S'^{\mathbf{SL}}_{in}(v_1, \ldots, v_p) = \{(i, in) \mid \forall j \in [1, p].\ (i, in) \in v_j \}$$

When calculating the individuals satisfying the out-node conditions only those that are recorded as *in* are taken to fail:

$$S'^{\mathbf{SL}}_{out}(v_1, \ldots, v_p) = \{(i, in) \mid \exists j \in [1, p].\ (i, in) \in v_j \}$$

This eliminates all pairs of the form (i, *out*) from the construction of interpretations.[13]

In this approach to dealing with variables, although the default schema is represented in the network thus enabling operations to be performed on the network without the need for instantiating particular instances of the default, e.g. looking for support for nodes or checking entailment relations, the information about individuals and the properties they possess are hidden in the dependencies and the valuations. If the fact that Tweety is a bird is asserted by a justification $\longrightarrow_{Tweety} B(x)$, then to assert that Tweety being a bird entails Tweety is my favourite bird (an expression applying to a single individual) it is necessary to restrict the set of birds (represented by individual/value pairs in the interpretation) to just Tweety.

This motivates a desire to be able to mix propositional and quantified representations within a single network. In order to do this, there is a need to have nodes n and m representing "Bird(Tweety)" and "Bird(x)" and have a mechanism for linking the values of the two so that $V(n) = v$ iff $(n, v) \in V(m)$. There isn't space in the thesis to detail this mechanism but there would be no problem in constructing such a link.

5.3 Rule-Based Expert Systems and IDNs

Rule-based expert systems are the epitome of AI success in terms of the number and scope of systems and their commercial success. They are based on the simple premise that knowledge can be easily and accurately formalised and represented by a collection of IF-THEN (IT) rules. That is, rules of inference that sanction the assertion of the consequent of the rule if the antecedent conditions are satisfied. Examples from renowned systems include:

> "IF the gramstain is gramnegative and the morphology of the organism is rod and aerobicity is anaerobic THEN there is suggestive evidence that the identity of the organism is bacteroides." [Shortliffe 1974, Buchanan and Shortliffe 1984];

[13] In this particular case given (i, *out*) never occurs, the pair (i, *in*) could simply be represented by the value i. However this technique of indexing (i.e. using simple individuals rather than individual/value pairs) is limited to those IDNs with two values. The more general technique may be applied to any IDN to maintain multiple simultaneous valuations.

"IF plagioclase has been altered to albite or minor sericite and plagioclase has not been altered to major epidote THEN the lateration of plagioclase is indicative of the barren-core zone." [Duda et al 1979];

"IF the most current active component is assigning a power supply and a UNIBUS adaptor has been put in the cabinet and the position it occupies in the cabinet (its nexus) is known and there is space available in the cabinet for a power supply for that nexus and there is an available power supply and there is no H7101 regulator available THEN add an H7101 regulator to the order." [McDermott 1980].

It cannot be claimed that IDNs are a general framework for capturing AI belief revision mechanisms if they cannot be used to capture the rules of belief revision represented by RBSs. Unfortunately, as I show in the following section, IDNs cannot do this without a substantial modification to the RBS philosophy and methodology.

The representation of RBSs as IDNs represents an advance, and is not the poor result that it might seem, for I show (in §5.3.2) that in order to produce disciplined and consistent belief revision in an RBS it is necessary to use a *defeasible* RBS. It is this concept of *defeasibility* that is at odds with existing RBS behaviour and it is this concept that allows RBSs to be presented as IDNs.

In §5.3.3 I outline the properties of defeasible RBSs, pointing out similarities with existing work, and in §5.3.4 I show how IDNs can be used to create such systems. §5.3.5 outlines the benefits of such an approach while §5.3.6 sketches some extensions. Finally §5.3.7 shows how uncertainty measures, one of the main justifications for using RBSs, can be integrated into IDN-based RBSs.

5.3.1 Rules and Dependencies

IF-THEN (IT) rules are not logical implications to be used in conjunction with an inference rule such as Modus Ponens (A, A→ B |– B), nor are they rules for the syntactic manipulation of symbols. They are "real" rules of inference (in the sense used by Israel [1980]) that are involved in the revision of belief and have been constructed from an expert's knowledge about what constitutes rational or coherent or useful judgements in a particular situation.

One of the things that distinguishes IT rules from logical implications is that they can only be driven in one way: satisfying the antecedent can lead to a belief in the conclusion but a lack of belief in the conclusion (or a belief in the negation of the conclusion) cannot be used to derive the negation of the antecedent condition. In this respect they are similar to dependencies in an IDN whereby support is one way and changes propagate forward[14]. Where IT rules and dependencies do differ is in the manner of their execution.

The standard OPS5 architecture for a simple RBS is a knowledge base consisting of information (facts, goals, assumptions etc) held in short term working memory (WM) and a set of rules (RB) held in long term memory combined with an inference engine. The system is driven by a match-resolve-execute cycle:

match rules whose antecedent condition is satisfied by the information in WM are stored as the *conflict set*[15]

resolve a rule(s) is selected from CS as the next rule to execute on the basis of some heuristic or control strategy (e.g. [McDermott and Forgy 1978])

execute the rule chosen by the conflict resolution strategy is executed with its consequent being added to WM

After a rule has been fired there is no connection between the antecedents and the consequent and although subsequent changes to WM may mean the antecedents are no longer satisfied, the status of the conclusion remains unaltered. This persistence of information is a product of the philosophy of RBSs producing solutions through search: the information in WM at any time is a particular problem solving state and the rules that apply in that state are the operators for moving to new states. As such the RBS is using a coherence approach to belief revision - it does not matter how information has come to be in WM (excepting information used in making control decisions) as long as the problem solving state is consistent.

[14] Obviously rules can be used in the opposite direction given goal-directed behaviour or backward chaining. However backward chaining does not lead to the assertion of facts but is a way of collecting information or constructing chains of inference that when driven in a forward direction will generate the desired conclusion.

[15] Rules may be added to an existing conflict set - a single structure may be used to store the conflict set, in which case this structure is updated rather than created during each iteration (e.g. the join network for RETE algorithms).

5.3.2 Non-Monotonic and Defeasible Inference

A consistency approach to belief revision in RBSs based on a "fire and forget" policy has two drawbacks. First, in domains where information is changing (e.g. the Ventilator Management system [Fagin et al 1979]), rules have to be continually re-run to update the conclusions and check their validity[16]. This causes a problem when updates (produced by external i.e. non-rule-driven events) are not commutative with rule firing. Adding the same information to working memory at different times can produce different results.

If information is given to the system in a phased manner (i.e. interspersed with the firing of rules) the result may differ from that produced when all the information is presented before any rules fire. For example, consider the rules:

(R1) If a and b THEN delete a

(R2) IF a THEN add b

(R3) IF a and b THEN add c

If the initial WM_0 = {a} and the conflict resolution fires each rule in turn as they are satisfied, then R2 fires producing WM = {a, b}, causing R3 to fire followed by R1:

Figure 5.3.2 (a)

resulting in WM_3 = {b, c}. The subsequent phased addition of {b} to WM_3 produces no change to the final result {b, c}. Yet executing the rule-base on WM'_0 = {a, b} (i.e. changing the timing of adding b) would result in R1 firing to produce a final state WM'_1 = {b}.

[16] This is not strictly true of all such systems. SOAR5 [Newell 1990] contains a TMS and does actually maintain dependencies between various working memory elements.

Secondly, it is impossible to represent and use rules of the form "if A and B then C" where B is a condition that is not instantaneous but has some temporal duration, e.g. "if the alarm has gone off and there is no evidence of a malfunction or another trigger event then there has been a burglary". In this rule, the conclusion is valid if the antecedents are valid at the time the rule fires, but remains valid only for as long as the lack of evidence persists.

It may seem that this type of rule is captured by non-monotonic logics and that some non-monotonic capability should be added to an RBS to perform this kind of inference. However, this claim should be examined closely with reference to what exactly constitutes non-monotonicity and with regard to the notion of "defeasibility".

In general, non-monotonic inference is defined to be a logical system (with some consequence relation $\mid\!\sim$) in which the addition of information prevents the derivation of other formulae which would have otherwise been derivable, i.e.

$$\Gamma \mid\!\sim \alpha \text{ but } \Gamma \cup \Psi \not\mid\!\sim \alpha$$

Non-monotonic inferences are often characterised as tentative conclusions withdrawn in the face of contradictory evidence. Typically (![17]) they are given a reading of

"IF α_1 ,..., α_n are true and β_1 ,..., β_m are consistent THEN γ" **

where β_i is consistent if there is no evidence of $\neg\beta_i$.

It has been stated that defeasibility is the property that conclusions are tentative, are "capable of being defeated" [Ginsberg 1987], and can be withdrawn given new and/or better information. It seems unclear from the literature whether defeasibility actually applies to the inference rule or to the conclusion that is inferred. I would argue that it makes little sense to say that the inference rule is sound (i.e. produces valid conclusions and is not defeasible) when the conclusion has been defeated or retracted. In other words, I believe defeasibility to be a property of the inference rule rather than of the conclusion.

To see this, imagine we were to say that a rule "If X then Y" is not defeasible, i.e. is not a tentative rule, but produces defeasible (tentative) conclusions. If we had some reason to

[17] For the uninitiated, this is a joke!!

disregard the tentative conclusion Y, we cannot do it simply by removing Y. If we were to do this, there is no reason that the rule could not be reapplied thus rederiving Y. The only ways to prevent this are: not let the rule be reapplied; invalidate or defeat the antecedent (X) of the rule; or remove or invalidate the rule itself. All these courses of action involve the rule itself. It is not possible to tell from a proposition itself, in isolation, whether or not it is defeasible.

Taking defeasibility to be a property of inference rules, the non-monotonic inference (**) above is defeasible, i.e. can be undone, in that the derivation of $\neg\beta_i$ prevents the derivation of γ. The defeasibility arises because the notion of consistency has a temporal duration - the rule does not require β_i to be consistent with the information derived at the time the inference is made, but after **all** possible inferences have been made.

The terms "defeasible" and "non-monotonic" are not synonymous for it is possible that non-monotonic inferences are not defeasible and that monotonic inferences are defeasible. I claim that if expert systems are to be used to capture rules such as the one above, then the system used should be one based on defeasible inferences, rather than on non-monotonic reasoning. In the following section I shall show how a simple RBS can be non-monotonic but not defeasible, and show why such a system should not be used to capture the above rule. I will then show how existing non-monotonic RBSs capture this rule in virtue of their defeasibility rather than their non-monotonicity. Finally I shall demonstrate how IDNs perfectly capture the notion of defeasible inferences and can be used to implement a defeasible RBS.

5.3.3 Defeasible Propositional Rule-Based Systems

The simplest rule-based system is one based on a propositional language $\mathcal{L}_P$. Rules $L \longrightarrow R$ consist of a left hand side (antecedent condition) $L \subseteq \mathcal{L}_P$ and a right hand side (consequent actions) $R \subseteq \{ +, - \} \times \mathcal{L}_P$. L represents the set of propositions that must be present in working memory (WM) for the rule to fire. When the rule is fired the actions in R are executed with the set of propositions $R^+ = \{ \alpha \mid (+, \alpha) \in R \}$ being added to WM and the set $R^- = \{ \alpha \mid (-, \alpha) \in R \}$ being removed from WM.

A rule-based system that allows for the deletion of elements in WM can be considered non-monotonic. Consider the closure of a knowledge base KB = RB ∪ WM to be the recursive application of all rules according to some conflict resolution strategy defined by a function *resolve* which takes a set of rules and returns a single rule to be fired. Assuming a static rule-base (this is only for brevity's sake), the closure of KB is equivalent to the closure of WM w.r.t. RB which in turn is given by:

$close_0(WM) = WM$

$close_i(WM) = (close_{i-1}(WM) \cup R_i^+) \setminus R_i^-$

for $r_i = L_i \longrightarrow R_i$ and $r_i = resolve(\{ r_j \in RB \mid L_j \subseteq WM\})$

$close(WM) = close_n(WM)$ s.t.

$close_n(WM) = close_{n+1}(WM)$ and $\nexists m<n.\ close_m(WM) = close_{m+1}(WM)$.

The closure of WM can be viewed as the set of consequences of WM with WM $\vdash \alpha$ iff $\alpha \in close(WM)$. The definition of non-monotonicity then translates into the condition:

$\alpha \in close(WM)$ but

$\alpha \notin close(WM')$ for $WM \subseteq WM'$

Obviously, given a rule $r = \{a, b\} \longrightarrow \{(-, a)\}$ and working memory WM = {a} then WM $\vdash$ a but WM ∪ {b} $\not\vdash$ a.

However, although a rule-based system allowing deletions (as outlined above) is non-monotonic, if it is based on a "fire and forget" model of execution, it is not defeasible. Although a rule (R2) may remove propositions previously added by another rule (R1) this deletion does not necessarily invalidate the original rule (R1). It may be the case that the original evidence that led to R1 asserting the conclusion is still present and removing the conclusion without changing this evidence can lead to situations where contradictory rules are continually asserting and retracting the same piece of information in turn.

Despite the confusion over equating non-monotonicity with defeasability (e.g. Rich [1983 p178], and Schaefer et al [1986 p918] are guilty of this) the concept of defeasibility is applicable to monotonic rules as well as non-monotonic rules. Consider a rule of the form IF A and B THEN C (as in §5.3.1), where B is a condition with temporal duration that is invalidated by the removal of information (rather than the addition of information as is the

case in §5.3.1). This rule is not non-monotonic for it is the removal rather than the addition of information which leads to the retraction of the conclusion. It is possible to have a retraction rule so that if B no longer holds then C is retracted but this introduces yet another problem: if C has been asserted by some other rule then the failure of B leading to the retraction of C would result in C being removed when in theory it should still be present.

The desire for defeasible and/or non-monotonic RBSs has led to the development of systems that have an augmented rule structure of the form "IF-THEN-AS.LONG.AS" [Schaefer et al 1986] and "IF-THEN-UNLESS" [Reinfrank et al 1986]. By their nature, these systems have had to maintain dependency records between the AS.LONG.AS or UNLESS conditions and the THEN conclusions. This in turn generates a need to record other conclusions ultimately supported by the augmented rules. This has been done either through the explicit use of a TMS [Reinfrank et al 1986] or through TMS-like functionality [Schaefer et al]. It is interesting to note that the TMS provides defeasibility and it is only the use of the closed world assumption in defining negation, or some other non-monotonic operator, that makes defeasible inferences non-monotonic. "AS.LONG.AS" conditions capture defeasibility while it is the absence of information referred to in "UNLESS" conditions that makes such inferences non-monotonic.

As an aside, if a defeasible RBS is wanted and it is undesirable that evidence can be overwritten but should rather be invalidated or withdrawn, the obvious way to do this is with a system that does not allow deletions or negative support but provides only positive support. Negative support for a proposition, e.g. α is false, should be translated into positive support for some other proposition e.g. $\neg\alpha$ is true, and some procedure used to resolve conflicts between the two where necessary.

In the rest of this section I shall examine some problems that arise in the interfacing of TMS and RBS. In particular I wish to examine how RBSs might be directly represented using dependency networks thus challenging the following:

> "It is not clear exactly how the validity maintenance abilities of the human expert can be naturally expressed in terms of such [dependency] networks. It may be possible for the expert to indicate how the individual working-memory elements [nodes] in the

> network are dependent on each other. However, the implementation-level network is at a considerably lower level of representation than the 'knowledge level' expert production rules. It would be unsatisfactory to require the expert to think about the domain at both levels." [Schaefer et al 1986, p919]

I would claim that in the case of defeasible inferences the process of specifying rules is isomorphic to that of indicating dependencies.

5.3.4 RBSs and IDNs

In a simple propositional system a rule of the form "IF α_1 ,..., α_n THEN β" can be represented by a monotonic SL-dependency $\{\alpha_1, ..., \alpha_n\}_{in} \longrightarrow_{SL} \beta$. This rule is defeasible, by the nature of IDNs. When all the antecedents are *in* the rule will "fire" and the value assigned to the consequent will be *in*[18]. In this case the rule is valid and the antecedents are satisfied. If any α_i is *out* then the rule does not "fire", i.e. the rule does not support β being *in* and unless another rule provides support then β will be *out*. In this case the rule is invalid and the antecedents are not satisfied.

A change in valuation that results in previously unsatisfied antecedents becoming satisfied or vice versa will lead to a corresponding change in value for the consequent if the value of that rule is sufficient to determine the value of the consequent (i.e. there are no other rules with the same consequent or all other rules that share this consequent are invalid).

Non-monotonic defeasible rules can be captured by a non-monotonic JTMS-IDN where lack of evidence is represented by non-monotonic links. E.g. the rule

"IF α_1 ,..., α_n and α_{n+1} ,..., α_m are consistent THEN β"

is represented by $\{\alpha_1, ..., \alpha_n\}_{in}\ \{\alpha_{n+1}, ..., \alpha_m\}_{out} \longrightarrow_{SL} \beta$.

Propositional rules in this form can be captured by existing TMSs or TMS-RBS combinations, so other than expounding the difference between defeasible and non-monotonic inferences, what does the study of IDNs add to existing work? A defeasible RBS

[18] This is of course a procedural notion of changes to an admissible valuation and relies on an update algorithm that is sound and complete. Denotationally the consequent will be *in* in any admissible valuation in which all the antecedents are *in*.

that relies on a stand-alone or separate TMS component, e.g. CAPRI [Reinfrank et al 1986, Freitag and Reinfrank 1987], suffers from three problems: duplication of information; the need to exchange information; and unwanted interactions caused by the exchange of information.

Taking the last problem first, consider the following set of rules [Reinfrank et al 1986 p27]:

(R1) IF p UNLESS q THEN r

(R2) IF p UNLESS r THEN q

(R3) IF r THEN q

The RBS is responsible for checking the value of the antecedents stored in the TMS, and if a rule antecedent is satisfied then the rule is fired and an appropriate dependency added to the TMS which then updates its valuation. Given V(p) = *in* (by assertion $\longrightarrow_{SL}$p) and a conflict resolution strategy that fires each valid rule in turn, R1 fires adding $p_{in}q_{out}\longrightarrow_{SL}r$, giving a valuation V = {(p *in*) (q *out*) (r *in*)} causing R3 to fire, adding $r_{in}\longrightarrow_{SL}q$. This creates an odd loop and the TMS is unable to create an admissible valuation. If R2 were already represented in the network by $p_{in}r_{out}\longrightarrow_{SL}q$ then it would be possible to "stabilise" the odd loop and construct an admissible valuation V′ = {(p *in*) (q *in*) (r *out*)} but because it is not possible to fire rules during the valuation construction phase stabilisation does not occur. In general the conflict resolution strategy and/or rule firing order that is employed by the RBS may not be reflected in the TM procedure used by the TMS.

The other two problems of duplication and transfer of information, being computational issues, are minor in comparison to the conceptual problem above. Firstly there is a duplication of the information in the rules and the dependency network. As rules fire, an image of the rule-base is built up in the TMS. Secondly when a rule fires the satisfiability check of the antecedents is mirrored by the TM process when dependencies are added to the rule base. Initially the checking at the TMS level is unnecessary while subsequently the checking at the RBS level is redundant, given the rule has already been instantiated in the TMS.

All these problems point to a solution of moving execution or firing of rules from the RBS to the TMS. Rather than storing rules in an RBS this component can be discarded (although see §5.3.6 and §5.2.4 for a discussion on pattern matching and variables) and rules coded directly as dependencies. However, it should be stressed that this type of system can only be used to represent defeasible rules. For procedural "fire and forget" rules this type of system is inappropriate.

5.3.5 Benefits of an RBS-IDN System

The explicit representation of rules as dependencies within an RBS-IDN system means that the problems listed above can be resolved. In particular (non-monotonic) defeasible inferences with automatic, incremental updating can be implemented without duplicating information nor having the interaction between RBS and TMS produce undesirable results. This gives the capability to deal with changing information, both assertions and retractions, in a clear, systematic and efficient manner.

In addition to resolving problems with existing systems there are additional benefits of using an RBS-IDN. These fall into two main categories: those that come from the way a particular admissible valuation is constructed; and those that come from changing the interpretation structure itself, i.e. what constitutes an admissible valuation.

The basic method of constructing admissible valuations as outlined in Chapter 4 is based on a depth-first approach that contains the potential for a parallel model of execution. In any given iteration in the interpretation process, nodes at particular depth have their values calculated. As all nodes at the same depth cannot depend on each other, the rules supporting those nodes do not interact and can be fired simultaneously.

This approach is conservative in timing. The execution of a rule is delayed until all rules sharing the same consequent can have their antecedent conditions checked. E.g.

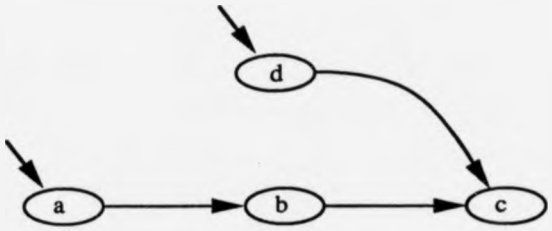


Figure 5.3.5 (a)

In the first iteration, values for a and d are calculated and although the value for $d_{in} \longrightarrow_{SL} c$ can be calculated, the rule does not "fire" thus calculating a value for c, until a value has been calculated for b. By manipulating the queue of node values to calculate (TC), and by exploiting redundancy[19] or assuming *nil* values for as yet uncalculated values, it is possible construct an interpretation in a more opportunistic manner corresponding to a depth-first approach.

Changing the order in which the admissible valuation (AV) is calculated through manipulating TC will not produce any different AVs for an unambiguous network[20]. However it should be noted that although values produced by using redundancy are not subject to revision, those values produced using assumed *nil* values may be recalculated as other nodes are calculated, and can only be thought of as tentative conclusions. This is not necessarily a disadvantage for it enables a tentative partial solution to be constructed quickly. This technique can be extended so that arbitrary values are assumed in a depth-first construction. The assumed values can be collected and become conditional assumptions that must be discharged (as in natural deduction systems [Lemmon 1965]) before values

[19] I.e. cases where a single dependency value is sufficient to determine a node's value e.g. if V(d) = *in* then V(c) = *in* for any admissible valuation V irrespective of V(b).

[20] If a network is ambiguous, i.e. $|VA(<N,D>)| > 1$ and the disambiguation relies on some procedural notion of how the interpretation is being constructed then different methods of construction may actually result in different AVs being produced.

supported by them can be considered part of an AV.

Procedural notions of conflict resolution in an RBS that determine how rules interact and which may result in different WM states, must be captured by explicit dependencies between rules in an RBS-IDN. This is an added advantage, for by forcing these interactions to be stated explicitly, the control structure is brought to the fore (as advocated by Georgeff [1982]). Because of the defeasible nature of the rules this control structure can be built through the use of an additional "rule identifier" antecedent. Thus the rules

(R1) IF p UNLESS q THEN r

(R2) IF p UNLESS r THEN q

become dependencies $\{p\ R1\}_{in} q_{out} \longrightarrow_{SL} r$ and $\{p\ R2\}_{in} r_{out} \longrightarrow_{SL} q$, where nodes R1 and R2 respectively mean R1 and R2 are executable or applicable in the current context. Asserting both rules are applicable, by adding $\longrightarrow_{SL} R1$ and $\longrightarrow_{SL} R2$, results in an ambiguous network and some external disambiguation procedure is necessary to decide which rule should fire. By removing $\longrightarrow_{SL} R2$ and adding $R1_{out} \longrightarrow R2$, an ordering is placed on the rules so that R2 is allowed to fire only if R1 is not.

This kind of explicit approach to conflict resolution is open to the kind of criticism levelled against R1/XCON [McDermott 1982]. That is, by mixing strategic or meta-level problem solving information (i.e. information saying when to use which rules) with object level rules, the knowledge base is no longer epistmeologically clean. In otherwords, the representation of some domain by facts, rules heuristics etc is diluted with information that is needed to make a particular system and/or representation scheme behave in a specific way. This information may be used to make the system chose one answer in preference to another, or to make the search more efficient but this information may have no use (or even worse, no validity) outside the particular system.

This criticism can be partly avoided by making the control information easily separable from the domain information. In the case of the IDN scheme proposed above, the use of rule-identifying antecedents means that although the control information is expressed in the same way as domain information (i.e. the system is mono-lingual - the meta-level and object level languages are the same), it is distinct. However, it should be noted that particular care

must be taken in specifying contradictions in an IDN that uses dependency directed backtracking. If contradictions have no epistemological basis independent of the systems behaviour, i.e. they are only included to make the search behave in a particular manner, then the criticisms of R1/XCON once again apply.

In addition to having multiple ways of driving the RBS-IDN forward, it is possible to use DDB to generate query-driven behaviour (see §4.3.2). If a certain set S of nodes represents solutions and another set A represents "askable" data (e.g. Mycins' LABDATA [Buchanan and Shortliffe 1984, p 64]) then if no solution is currently supported then backtracking to nodes in A provides a way of focusing data gathering. Heuristics such as maximum discrimination (choose nodes that support the smallest and undermine the greatest number of solutions), maximum impact (a minimum number of additional nodes must be established for a solution to hold), and depth (closeness to original causes or askable data, length of reasoning chain), provide a way of choosing which questions to ask first. Similarly, if some quantitative measure of uncertainty is being used (see below) and no solution exceeds a certain minimum value then backtracking from the highest ranked solution may establish that solution with the required degree of certainty.

The interpretation structure (i.e. the actual values calculated and the summation functions used to do this) itself can also be changed easily enabling the same rule-base to be used for a variety of different tasks. In particular, given the static nature of an AV (i.e. rules that are fired support their consequents at all times) it is easy to implement a consistency checking or verification scheme. COVADIS [Rousset 1988] assigns ATMS-style labels to elements in a rule-base and uses these values to make modifications to the rule base. Given the inter-changability of J- and NATMS interpretation schemes (see §5.1) it is possible to check the consistency of a network within the same framework as it is executed, using the same data structures and without the need for external programs, or the need to simulate execution of the rule-base in every conceivable situation. The deterministic and declarative nature of rule firing also makes it possible to investigate entailment relationships between propositions and rules. For example, given

$\{a\ b\ R1\}_{in} \longrightarrow_{SL} c$

$\{c\ R2\}_{in} d_{out} \longrightarrow_{SL} e$

$\{f\ R3\}_{in} \longrightarrow_{SL} e$

$\{g\ R4\}_{in} \longrightarrow_{SL} d$

it can be shown that there is no relationship between c and f but that g being *out* is a necessary condition for R2 to fire and that unless R3 is fired or e is asserted, if g is *in* then e will be *out*.

The static nature of the interpretation w.r.t. the network also means that explanations can be generated by examination of the network. It is not necessary to maintain a rule trace which, in the presense of non-monotonic rules, may be less than clear. For example, given rules

(R1) IF α and β is unknown THEN add γ

(R2) IF β THEN delete γ

(R3) IF δ THEN γ

and a working memory WM = $\{\alpha\ \delta\}$, then R1 fires adding γ. A later assertion of β would fire R2 (or retract R1 if the system were defeasible) leading to a retraction of γ followed by a firing of R3 and a second addition of α. A rule trace would contain all this information which must be searched to find the relevant information.

As shown in §5.2 it is also possible, through the use of ATMS-type indexing (as happens with KEE-Worlds™) to maintain multiple simultaneous scenarios. Finally, as shown in the next section, it is possible to combine defeasible non-monotonic inferences with numerical measures of uncertainty in the context of RBSs.

5.3.6 Extensions to Propositional RBS-IDN

Propositional RBSs are powerful enough to capture enough information to perform useful problem solving tasks, though obviously propositional logic does not have the expressiveness of first order logic. In particular propositional logic does not contain variables and quantifiers though it is capable of dealing with ground (variable-free) predicate

formulae. There is a trade-off to be made between tractability and expressiveness (a prime example being in frame-based systems), which in the case of RBS can be seen in the "Object-attribute-value" (OAV) representation scheme. This is one of the most popular forms of knowledge representations within RBSs and has been used in a large number of commercial tools and shells (e.g. M1, OPS5, S1, Expert-Ease ... [21])

An OAV representation is based on objects that have attached properties. These in turn can take a variety of values from {true, false}, integer or real numbers or an arbitrary set of symbols e.g. colours {red, green blue ...}. Antecedent conditions are conjunctions, disjunctions and negations of tests of an attribute's values where the tests are equality, less than, greater than or membership tests. Depending on the values and test used, the OAV representation is equivalent to FOL restricted to unary predicates, with equality, or more expressively (in terms of values and tests handled) binary and unary predicates, or most strongly unary functions with binary (and unary) predicates. In general rules can contain variables but no explicit quantifiers - any rule containing variables is considered to be universally quantified.

Considering OAV facts to be functions representing attributes applied to individuals representing objects that return values within the language means that all the knowledge is represented within the syntax of the language. However, by considering attributes to be predicates that apply to individuals with differing truth values means that some of the knowledge is moved to the interpretation structure. For example, Michalski et al [1976] (as reported in [Haack 1978]) have constructed a 12-value logic where the truth values are related to months of the year in which a proposition holds true. Rather than saying "Red spots appear in February" and assigning a value "true", the proposition is reduced to "Red spots appear" and a truth value of "2" is assigned meaning the proposition is true "in February". However the second rendering can be translated into the first without loss of information. Where the information is placed in the semantic/syntactic divide is not necessarily a matter of expressive power, but of computational issues.

[21] See [Harman and King 1985] for an overview of these, and many more, systems.

In the case of IDNs the syntax of propositions is hidden in the nodes and the semantic value assigned to a proposition is simply that assigned by the summation rules. The debate over where information should be placed translates into a debate about what interpretation mechanism should be used. Consider the fact "the colour of block11 is red". This is most intuitively captured as a unary function *colour* applied to object "block11" with resulting value "red" (*colour*(block11) = "red"), or as a two place predicate (colour(block11, red)). In either case the nodes representing these statements would be assigned a value in {*true*, *false*}. Alternatively a node could be used to represent the colour of block11 as a unary predicate 'colour(block11)' with a value being assigned from a set of values = {red, green, blue ..}. Combining these two different approaches to representing values with the two different approaches to representing individuals (in §5.2.4) as either information stored in the node or in the interpretation, gives four basic representation schemes for OAV systems.

S1: basic propositional scheme - instantiation by interface

S2: multi-valued scheme

S3: propositional with variables

S4: multi-valued with variables

Figure 5.3.6 (a)

Looking at each scheme in turn, the following benefits and drawbacks can be attributed. S1 involves a simple interpretation scheme. All the information is stored as propositions attached to nodes, and queries can be evaluated by look-ups on the node-proposition mapping and getting a single value from a current admissible valuation[22]. There are two major drawbacks to this approach: the need for pattern matching and node/dependency instantiation capability in the user interface; and the large number of nodes and dependencies produced. However, given the simple structure of the interpretation, these dependencies are fewer in type and of less complexity than with other systems.

Scheme S2 prevents the proliferation of nodes and dependencies by having a single node capable of representing all possible values of an attribute, rather than having a node for each separate value. This in turn allows a single dependency to be used to capture all the dependencies of one attribute's values on another. This results in a smaller network which can be further reduced in size by having conflicts in values resolved through the summation function assigning a single value, rather than using more nodes and dependencies to capture this resolution process. For example, consider the relationship between the colour and taste of fruit juices:

colour	**taste**
orange	orange
green	lemon and lime, lemon
yellow	lemon, lemon and lime
red	blackcurrent

[22] This assumes a fully instantiated network - in practice if all updates to the network have been made then if a node is not present in the network then there are no nodes even potentially capable of supporting the query and the CWA can be invoked to assign a *nil* value.

Representing these rules in S1 takes 8 nodes and 4 dependencies while in S2 this takes 2 nodes and 1 dependency. If values in S2 are sets of attribute values then specific combinations of tastes and colours could be allowed (green and yellow or lemon and lemon and lime) while others (e.g. green and red) are contradictory. To represent such conflicts in S1 would take more dependencies. The negative side of this approach is that it leads to a less uniform interpretation structure with a variety of different disjoint sets of values being assigned to different nodes. The moving of information to the interpretation mechanism and the resulting increase in complexity makes secondary tasks such as tracing support and determining entailment relationship more complex. There is also a decrease in modularity if a number of S1 dependencies are compiled into a single S2 dependency[23].

The individual pros and cons of S3 have been discussed in §5.2.4 and S4 inherits the problems and advantages of both S2 and S3.

In conclusion, S1 is to be preferred to S2 if the values that attributes take are discrete and few in number and there are no obvious correlations between values of attributes. As mentioned in §5.2.4, S1 is to be preferred to S3 if the universe of discourse is small, i.e. there are are few individual objects, or the rules are narrow in scope and apply to few individuals or ranges of values.

In the context of RBSs, it should be noted that the use of a pattern matching interface to instantiate dependencies is not the same as that of a pattern matcher firing rules. A dependency is instantiated whenever part of its antecedent condition is present (as opposed to firing when all the antecedent condition is satisfied) and there is no notion of conflict resolution nor any other procedural interactions between additions.

5.3.7 Uncertainty and the RBS-IDN

One of the main advantages cited for rule-based systems over a system based on logic or semantic networks is the ability to deal with uncertain information. That is, a numerical measure that represents how likely or how certain a piece of information is. Such a measure

[23] If the rule-base is stable (i.e. changes to the rules are not likely to occur) and homogeneous (i.e. have a similar status in terms of origin and/or confidence) then this is less of a problem.

can either be based on classical probabilities [Duda et al 1976], Dempster-Shafer theory [Ginsberg 1984], or certainty factors [Shortliffe 1976].

Systems based on numerical uncertainty measures can either be *extensional* in that numbers are assigned to individual propositions in the same way that truth values would be assigned in a particular interpretation, or *intensional* in that uncertainty is assigned to particular interpretations and within each interpretation a proposition is either true or false [Pearl 1988]. This classification is in the same vein as the procedural/declarative or syntactic/semantic distinctions [ibid p3]. However, given that probabilities can be used both extensionally and intensionally[24] and the same applies to Dempster Schafer theory, this distinction is less than useful in deciding what kind of system of uncertainty to use.

Pearl [1988, p6-12] himself argues against extensional systems for three reasons; bi-directional inferences are not handled properly; it is difficult to retract information; and by combining evidence locally the source of information is not considered and the same evidence can be repeatedly used to inflate the certainty of some proposition. Yet the system of Bayesian networks that he proposes appears to be a procedural or extensional use of probabilities.

Of the arguments that he puts forward, only the problem of combining evidence is justified with regard to an RBS-IDN. The problem of retracting information is handled by defeasible rules. The counter-intuitive results produced by examples of bi-directional inferences are the result of sloppy knowledge representation and of trying to handle deductive and abductive inferences in exactly the same way with the same mechanisms. The problem of multiple paths from one proposition leading to multiple support of a single common descendent is difficult to resolve and indeed Pearl does not do this himself. The basic method of local computation for propagating beliefs in a causal network only applies to trees or polytrees (i.e. *singly connected* networks where there is at most one path between any pair of nodes). When dealing with multiply-connected networks it is necessary to employ techniques that remove the multiple connection either through joining nodes together

[24] E.g. P(A) = 0.75 can be interpreted as meaning that A is true in 75% of cases (intensional) or that in any given situation A has a chance of being true three times out of four (extensional).

or effectively breaking links between nodes.

If one tries to combine an RBS using extensional uncertainty with a TMS to provide defeasible inferences, the problem of assigning values to nodes (see §6.3.5) arises. If the uncertainty information is stored within the node, i.e. nodes represent propositions such as "α has value 0.75", then when the value is changed a new node must be created along with new dependencies, and the structure of the rule-base becomes lost. It is only by representing uncertainty information in the interpretation that defeasible inferences using uncertainty can be handled. This motivates a desire for multi-valued TMSs, and IDNs provide the perfect vehicle for implementing such systems.

In the rest of this section I will look at how IDNs can be used to represent and deal with a variety of uncertainty measures that could be incorporated for use within an RBS. In §5.3.7.1 I will briefly point to work done on using the Dempster-Shafer theory of updating, before showing (in §5.3.7.2) in detail how a certainty factor model can be constructed. Finally in §5.3.7.3 I will show how probability can be similarly handled through work on Bayesian Networks.

5.3.7.1 Dempster-Shafer Theory and the ATMS

Many papers have been written on ways of providing support for the Dempster Shafer theory of belief revision through the use of multi-valued TMSs. These have focussed on the use of the ATMS to represent different worlds, and on using the labels generated by the TMS in conjunction with a distribution function over the base set of assumptions, to generate uncertainty values for propositions[25].

Being based on the ATMS, the application of DS theory in this way is obviously amenable to use within the IDN framework. Not only that but the flexibility of the IDN framework with regard to easily changing interpretation structures means that it is easy to implement and switch between a system that simply uses ATMS labels or ATMS-DS labels

[25] It is interesting to note that here values are being assigned to nodes on the basis of values assigned to particular worlds (i.e. intensionally), when the labels from which those values are calculated have in turn been generated from other nodes in the network (i.e. extensionally according to syntactic criteria).

or some other interpretation mechanism.

5.3.7.2 Certainty Factors

The *certainty factor* model is an attempt to circumvent the large amount of data (in the form of conditional probabilities) needed to build a usable statistical model in domains such as medical diagnosis. As such, it is an attempt to model subjective or quasi-probabilities, where p(h) is not the frequency of some event h but a measure of belief (typically ranging from 0 to 1) in h at some particular time.

The result of firing rules is to change the subjective probability of the consequent of the rule (h) through increasing or decreasing its belief given the certainty associated with the evidence (e) that caused the rule to fire. The *measure of increased belief* in h given e, MB(h, e), is a proportional decrease in the uncertainty associated with a belief in h (where the uncertainty is measured by how far p(h) is from certainty, i.e. 1), dependent on how much the subjective probability has increased in the light of new evidence:

$$\begin{aligned} MB(h, e) &= \frac{p(h \mid e) - p(h)}{1 - p(h)} && \text{for } 1 > p(h \mid e) > p(h) \\ &= 0 && \text{if } p(h \mid e) \leq p(h) \\ &= 1 && \text{if } p(h) = 1 \end{aligned}$$

Similarly the measure of increased disbelief in h given e, MD(h, e) is the proportional decrease in the uncertainty of disbelief (measured by how far p(h) is from 0):

$$\begin{aligned} MD(h, e) &= \frac{p(h \mid e) - p(h)}{p(h)} && \text{for } 0 < p(h \mid e) < p(h) \\ &= 0 && \text{if } p(h \mid e) \geq p(h) \\ &= 1 && \text{if } p(h) = 0 \end{aligned}$$

The measures of increase and decrease of belief in h given e can be combined into a single figure, the certainty factor CF(h, e) that summarises the size and direction of change in the subjective probability.

$$CF(h, e) = \frac{MB(h, e) - MD(h, e)}{1 - min\{\, MB(h, e), MD(h, e) \,\}}$$

This gives a range of values for CFs from −1 to 1.

In reality, all propositions can be viewed as having no subjective probability (or more accurately, no bias in the subjective probability) in the first instance. This is done by considering prior probabilities (e.g. an expert's preconceptions of the likelihood of some event) as pieces of evidence themselves. In this way the propagating and combining of pieces of evidence is reduced to manipulating changes in belief and disbelief. In early certainty factor models this was done by propagating and maintaining separate MBs and MDs and then combining them[26]. Later models simply used CFs.

For a hypothesis h_i and events or pieces of evidence e_j the following rules have been defined [Buchanan and Shortliffe 1984, p255] for combining certainty factors:

(R1) $CF(h, e_1 \text{ and } e_2) = CF(h, e_1) + CF(h, e_2) - \{CF(h, e_1) \times CF(h, e_2)\}$ for $CF(h, e_1), CF(h, e_2) \geq 0$[27]

$= CF(h, e_1) + CF(h, e_2) + \{CF(h, e_1) \times CF(h, e_2)$ for $CF(h, e_1), CF(h, e_2) \leq 0$

$= \dfrac{CF(h, e_1) + CF(h, e_2)}{1 - min\{\,|CF(h, e_1)|, |CF(h, e_2)|\,\}}$ otherwise

(R2) $CF(h_1 \text{ and } h_2, e) = min\{\, CF(h_1, e), CF(h_2, e)\,\}$

(R3) $CF(h_1 \text{ or } h_2, e) = max\{\, CF(h_1, e), CF(h_2, e)\,\}$

(R4) $CF(h, s) = CF'(h, s) \times CF(s, e)$

Rule 4 is designed to model a priori values associated with pieces of evidence. CF'(h, s) is the certainty value of h when s is known with certainty, i.e. CF(h, e) = 1, but when s is uncertain, certainty in h given s, i.e. the current value CF(h, s), is factored accordingly.

Given a rule in the knowledge base, $r = a \wedge b \rightarrow c$, the CF combination rules above can be composed to give a single CF value for the support provided by a and b, through r, for c:

$$CF(c, r) = CF(r, e) \times min\{CF(a, e), CF(b, e)\}$$

Considering rules as justifications, R2 and R3 dictate how antecedent values are combined to

[26] Certainty factors were derived using the equation CF(h, e) = MB(h, e) – MD(h ,e).

[27] Note the similarity with $P(AB) = P(A) + P(B) - P(A \cap B) = P(A) + P(B) - \{P(A) \times P(B)\}$ when A and B are independent. This points to one of the drawbacks of CFs mentioned later: it operates on an implicit assumption of independence between events.

get a single value for a rule; R4 shows how CFs can be attached to rules in order to reflect relative strengths of rules; and R1 defines how values associated with rules can be combined into a single value for a proposition. The rules above are associative and can therefore be generalised and applied to any number of values.

Considering conjunctive monotonic rules $a_1 \wedge ... \wedge a_n \rightarrow e$ as dependencies in an IDN, a corresponding set of monotonic dependencies $R_\wedge$ can be defined:

$$R_\wedge: (a_1, ..., a_n)_+ \; b_r \longrightarrow_\wedge e$$

Positive (+) antecedents represent the conjunctive antecedents of the rule and an extra class of antecedent (r) in added to indicate the rule strength. The following summation functions are then defined:

$$S^{\wedge}_{\wedge,+}(v_1, ..., v_n) = min\{v_i \mid 1 \le i \le n\}$$
$$S^{\wedge}_{\wedge,r}(v_1) = v_1$$
$$S^{\mathbf{D}}_{\wedge}(v_+, v_r) = max\{v_+, 0\} \times v_r$$

The function for combining values for dependencies is a generalisation of R1 presented above:

$$S^{\mathbf{N}}(v_1, ..., v_n) = 1 - (1 - v_1) \times ... \times (1 - v_n) \quad \text{when } v_i \ge o \text{ for all } i$$
$$= -S^{\mathbf{N}}(-v_1, ..., -v_n) \quad \text{when } v_i \le o \text{ for all } i$$
$$= \frac{S^{\mathbf{N}}(w_1, ..., w_m) + (x_1, ..., x_p)}{1 - min\{|S^{\mathbf{N}}(w_1, ..., w_m)|, |(x_1, ..., x_p)|\}} \quad \text{otherwise}$$

where $\forall 1 \le j \le m.\ w_j \in \{ v_i \mid 1 \le i \le n,\ v_i \ge 0 \}$ and $\forall 1 \le k \le p.\ x_k \in \{ v_i \mid 1 \le i \le n,\ v_i \le 0 \}$

Disconfirming rules, i.e. rules that decrease the belief in a proposition, are handled by using a negative CF attached to the rule.

Negation of antecedents can be handled by using an additional class of *negative* (–) antecedents. The value of a negated proposition is obtained simply by changing its sign. For example, if π has a certainty factor of –0.6, i.e. is believed to be false to a degree 0.6, then $\neg\pi$ will be believed to be true with a degree 0.6. The summation function for negative antecedents is changed (from that of positive antecedents) to reflect this switch in sign. The summation for the dependency is changed to include the extra class of antecedent and to take

the minimum level of support of all positive and negative antecedents.

$$S^{A}_{\wedge,-}(v_1, ..., v_n) = S^{A}_{\wedge,+}(-v_1, ..., -v_n)$$

$$S^{D}_{\wedge}(v_+, v_-, v_r) = max\{ min\{v_+, v_-\}, 0\} \times v_r$$

Defeasible non-monotonicity can be introduced by using yet another class of antecedent, *uncertain* (˜) antecedents, where ˜π reads "if π is unknown then ...". The value associated with an uncertain antecedent is proportional to its closeness to 0, and the smallest value for uncertain nodes contributes to the value of the rules in the same way as negative antecedents were included.

$$S^{A}_{\wedge}\text{-}(v_1, ..., v_n) = min\{ 1 - |v_i| \mid 1 \leq i \leq n\}$$

$$S^{D}_{\wedge}(v_+, v_-, v_r) = max\{ min\{v_+, v_-, v\text{-}\}, 0\} \times v_r$$

If π has a certainty factor of 0, i.e. there is no evidence supporting the truth or falsity of π or the evidence for and against π is evenly balanced, then π is unknown with certainty.

Ignoring the philosophical and technical arguments against CFs for the moment, there is a practical problem associated with the rules for combining CFs. This concerns the treatment of disjunctive rules.

In Mycin the interpretation of disjunctive antecedents is handled by R3:

$$CF(h_1 \text{ or } h_2, e) = max\{ CF(h_1, e), CF(h_2, e) \}$$

Consider the case where: h_1 is supported by e_1, i.e. $CF(h_1, e_1) > 0$; h_2 is supported by e_2, $CF(h_1, e_1) > 0$; and e_1 has no effect on h_2 nor e_2 on h_1, i.e. $CF(h_2, e_1) = CF(h_1, e_2) = 0$. Applying R1, we get the following value for h_1 given e_1 and e_2:

$$CF(h_1, e_1 \text{ and } e_2) = CF(h_1, e_1) + CF(h_1, e_2) - \{CF(h_1, e_1) \times CF(h_1, e_2)\}$$
$$= CF(h_1, e_1)$$

and a similar value for h_2 given e_1 and e_2:

$$CF(h_2, e_1 \text{ and } e_2) = CF(h_2, e_2).$$

Applying R3 (the disjunctive rule for hypotheses) to get a value for h_1 or h_2 given e_1 and e_2 produces the following:

(C1) $CF(h_1 \text{ or } h_2, e_1 \text{ and } e_2) = max\{ CF(h_1, e_1 \text{ and } e_2), CF(h_2, e_1 \text{ and } e_2) \}$
$= max\{ CF(h_1, e_1), CF(h_2, e_2) \}$

The problem arises if R1 is applied before R3:

(C2) $CF(h_1 \text{ or } h_2, e_1 \text{ and } e_2) = CF(h_1 \text{ or } h_2, e_1) + CF(h_1 \text{ or } h_2, e_2)$
$- \{CF(h_1 \text{ or } h_2, e_1) \times CF(h_1 \text{ or } h_2, e_2)\}$
$= CF(h_1, e_1) + CF(h_2, e_2) - \{CF(h_1, e_1) \times CF(h_2, e_2)\}.$

The composition of the rules R1 and R3 is not commutative and unless there is a set way of applying rules there will not be a standard way of modelling this situation. Intuitively, C1 appears to model the situation where h_1 and h_2 are disjoint, i.e. the disjunction is exclusive - the maximum support for the disjunction being the maximum support for either disjunct.

The second case (C2) appears to model an inclusive disjunction and is easily modelled in an IDN giving the summation rules above:

$e_1 \longrightarrow_{\text{and}} h_1$, $e_2 \longrightarrow_{\text{and}} h_2$, $h_1 \longrightarrow_{\text{and}} h1 \text{ or } h2$, $h_2 \longrightarrow_{\text{and}} h1 \text{ or } h2$

The exclusive "or" case can only be modelled by the inclusion of a new type of dependency with disjunctive antecedents:

$R_{\text{or}} : (a_1, \ldots, a_n)_+ \; b_r \longrightarrow_{\text{or}} e$

and summation functions:

$S^{A}_{\text{or},+}(v_1, \ldots, v_n) = max\{v_i \mid 1 \le i \le n\}$
$S^{A}_{\text{or},r}(v_1) = v_1$
$S^{D}_{\text{or}}(v_+, v_r) = max\{v_+, 0\} \times v_r$

This example (of ambiguity in an existing representation) demonstrates a general problem that IDNs will help to resolve. By attempting to code CFs as an interpretation mechanism, an ambiguity in the CF representation has been resolved and the representation extended, resulting in a more expressive system.

A larger problem that cannot be so simply resolved concerns the conceptual foundation of CFs. As noted earlier, CFs are supposed to capture pseudo probabilities. Given this, one

would also expect CFs to behave in a similar manner to probabilities, with the implication being that CF(h, e) approximates to P(h | e). Indeed, when there is confirming evidence e (i.e. MD(h, e) = 0) and p(h) is small,

$$CF(h, e) = MB(h, e) = \frac{P(h \mid e) - P(h)}{1 - P(h)}$$

and CF(h, e) will tend to P(h, e) as P(h) tends to 0. This however is only an extreme case and the approximation breaks down at higher values.

In the same way that CFs for propositions are supposed to behave like probabilities, the CFs associated with rules are supposed to act like a priori conditional probabilities. However, given the CFs reflect subjective judgements there tends to be little (external) justification beyond the fact that a set of rules produces the right result. There is also clustering of CFs around particular, functionally critical, points: e.g. 0.2 for rules that have small influence on a proposition, and 0.8 for (almost indicative) rules that have large influence. This implies that granularity in distinctions between the significance of two events is very small - either two events are significant (or insignificant) and cluster around 0.8 (or 0.2) with a very small difference between the two, or one is significant while the other is not producing a large distinction.

Finally, there is a deep, built-in assumption about the independence of evidence [Buchanan and Shortliffe 1984, p267]. This is an obvious outcome of designing a system that attempts to dispense with a large number of conditional probabilities. Although this independence is necessary to ensure qualitatively correct behaviour (for example, by not counting a piece of evidence more than once), there is no guarantee of independence in the structure of the rule-base.

These three points form the basis of arguments against the use of Certainty Factors: CFs do not have similar values to equivalent conditional probabilities; typically CFs need have no external justification and cluster at conceptually or functionally (in system terms) significant points; and there is no guarantee that the assumptions under which the CF model works will actually be built into any system that uses CFs.

There is no point in providing a CF-based system if no attempt is made to refute the arguments against CFs. There are three points I wish to make. The first is the standard defence of CFs: the CF model is an ad hoc approach and the validity of this approach can only be justified by its successful application in a particular domain. E.g. "The empirical success of MYCIN using the model of Buchanan and Shortliffe (i.e. CFs) stands in spite of theoretical objections" [Adams 1986]. Providing CFs within an IDN framework can be justified on the same basis and it is up to the designers and users of this tool to do so.

Having said this, the success of MYCIN is partly attributable to the independence of causes and symptoms in MYCIN's domain, and success in using CFs is likely to increase as independence between rules increases. By representing rules as a network in an IDN it is easy to enforce independence by checking for multiple paths from a single antecedent to a single conclusion.

Finally, as outlined in §6.5.2 it is possible to use a learning algorithm to adjust the CF values assigned to rules in order to minimise the discrepancy between system performance and expected results. In this way, the subjective probabilities are used as a starting point in order to produce a more accurate system, based on a set of test data. This avoids problems of tuning CFs for rules in isolation and on an ad hoc basis.

5.3.7.3 Bayesian Networks as IDNs

Attempts to combine the rule-based approach of certainty factors with a more statistically valid approach have led to the development of systems based on *Bayesian networks* [Pearl 1988].

In a Bayesian network, nodes are used to represent statistical variables and a belief value is assigned to each node x by a local computation. The belief value "Bel(x)" is a tuple consisting of probabilities assigned to each possible value of a variable. If the variable represented the ability to fly this might have two possible values, "yes" and "no", or four - "very well", "well", "badly" and "not at all". Bel(x) is calculated by summing the evidence available both from parent nodes (representing causal information) and from child nodes (diagnostic information) using the belief values assigned to these nodes. In particular given

a node x with n parents and m children

$$BEL(x) = \alpha * [\prod_{j=1}^{m} \mu_j(x)] * [\sum_{p \in P} P(x \mid p) \prod_{i=1}^{n} \pi(p_i)]$$

where $\mu_j(x)$ is the (posterior) probability distribution supported by the j-th child (excluding the influence of x); and where p is a permutation of the set of parental causal values, $\pi(p_i)$ is the (prior) probability distribution of the i-th parent value of that permutation, and $P(x \mid p)$ is the probability distribution of x conditioned on p. These distributions are combined using the outer product * (i.e. (a, b)*(c, d) = (a×c, b×d)) and the result is normalised using some constant α (so that the sum of probabilities for each possible value of x is one, $\prod_{i=1}^{r} x_i = 1$).

In effect each node has an associated prior (π) and posterior (μ) probability distribution calculated from its parent and child nodes respectively. As prior and posterior distributions are altered at leaf nodes (that represent initial causes and final outcomes), prior probabilities are propagated forward down the network while the posterior probabilities are propagated backward. The conditional probabilities associated with the casual links are similar in nature to certainty factors associated with rules and are a static part of the system entered during the knowledge ac quisition or model building phase.

By having separate dependencies to control the propagation of prior and posterior probabilities, and given the local nature of the belief value computation it is easy to see how Bayesian networks conform to an IDN model of computation. It is interesting to note that in the case of purely causal reasoning posterior distributions can be treated as uniform, i.e. $mu(x_i)$ for all values of all nodes[28], and backward dependencies and propagation are not required.

Because of the possible causal dependence between parents and the need to condition the posterior probability on the **joint** distribution of the parents, the parents must all be linked as a single dependency with an associated conditional probability matrix. This leads to a loss of modularity as causal links cannot be added without modifying the existing network. However, this is what one would expect as the associated conditional probability

[28] as with *anticipatory nodes* [Pearl 1988 p181]

matrix is bound to change (if only to be expanded) given the introduction of new variables. In special cases, if independence can be assumed for all parent variables and values then this is not the case. Causal links can be represented by individual dependencies and the j oint distribution can be calculated from the individual distributions for each parent.

In conclusion we can see that Bayesian networks can be represented as a particular form of IDN and that, as originally presented, Bayesian networks and IDNs share much of the same underlying philosophy. Finally, Bayesian networks show that it is possible to use extensional uncertainty information in a principled manner. This is one of the main thrusts of the research into IDNs - to provide principle and semantics to the use of potentially ad hoc network representations.

5.3.8 Conclusion

In this section it has been shown how the RBS paradigm must be adapted in order for the rules contained in such systems to be represented within an IDN framework. The benefits of doing this include: a generic ability to handle defeasible, non-monotonic inferences; and the ability to perform a variety of tasks using the same network structure simply by changing the interpretation mechanism. One such task (reasoning with uncertain information) is examined in depth and it is shown how various approaches can be implemented. All these approaches benefit from the fundamental characteristics of IDNs: a network representation is used to explicitly represent dependencies between data; values are calculated using only values from directly linked nodes; and changes in values are only possible through changes in the network (given an unambiguous network).

5.4 Inheritance Networks

Semantic networks or inheritance networks were one of the first AI knowledge representations and found favour because of their ease of understanding and intuitive nature. In their most simple forms such networks perform reasoning through simple network traversal algorithms.

In this section I show how a variety of these algorithms, with a variety of complexities, can be implemented as IDNs. In particular just a single network structure is required with different methods of inheritance implemented through different summation functions over different sets of values. Each set of summation functions effectively defines a different semantics for the same network.

Having shown how a variety of inheritance systems can be implemented, some theoretical results will be proved that relate to the constructibility and ambiguity of interpretations. This is an attempt to generalise Touretzky's work on the mathematics of inheritance systems, and show how IDNs provide a framework for the analysis of any network algorithm. In this respect, the work in this chapter in particular, and in the thesis as a whole, can be seen as an extension of Touretzky's ambitions: that is, to highlight problems with existing (network-based) algorithms and through a more formal approach to their specification, gain an ability to analyse and solve them.

5.4.1 Basic Network Representation

For the sake of this thesis I will consider a semantic network to be a labelled directed graph. The nodes of the graph represent classes of objects (or variously, sets, concepts, properties etc) and/or individual objects that may (or may not!) be members of some class (or classes) of objects. All the objects in a given class share the properties of that class (although see below for exceptions to this when considering inherited properties), and a predicate that denotes the members of that class can be constructed. The links between nodes represent relationships between classes and/or individuals and can be thought of as unary or binary predicates in FOL.

These relations can be of four different types: it can relate one concept or set of individuals to a second concept or set of individuals. For example, consider the following network:

Figure 5.4.1 (a)

This could be considered as one of four possible interpretations[29].

- $\forall x,y.\ canary(x) \wedge yellow(y) \rightarrow colour(x, y)$

  ≡ every canary and every yellow object have the same colour.

- $\forall x.\ canary(x) \rightarrow colour(x, yellow)$

  ≡ every canary is coloured yellow [30]

- $\forall y.\ yellow(y) \rightarrow colour(canaries, y)$

  ≡ every yellow object is of the same colour as the set of canaries

- colour(canaries, yellow)

  ≡ the colour of the set of canaries is yellow.

The last two of these links are less intuitive reading but could be thought of as relating properties of the set of canaries in abstract terms. For instance one might want to say that the set of canaries, or "the canary" referring to a single individual (concept) that embodies the set of canaries, is yellow without actually committing oneself to saying that each particular individual is indeed yellow.

Of the links or relations used in semantic networks, the most common and familiar type of link is the "IS-A" link. This link has been given many different interpretations: indeed, it has been said that "there are almost as many meanings for the IS-A link as there are knowledge representation systems" [Brachman 1983]. Of the ones given by Brachman [1983 p32], many of them correspond exactly to the type of distinctions made above. Of these interpretations, given that nodes stand for classes of objects, the appropriate interpretation is

[29] This is without even considering the possibility of links representing proto-typical or default relationships, or indeed links having any other different type of meaning (e.g. see [Woods 1975])

[30] This is equivalent to yellow(colour(x)) where colour is a partial function, i.e. each node has at most a single colour.

that of generalisation (or subset when considering the corresponding predicates)[31].

For example, consider Tweety the bird: bird IS-A (!) mammal and therefore the class of mammals includes all individuals that are birds. The predicate "mammal" is true for all individuals that satisfy the predicate "bird" and the IS-A link can be interpreted as a universally quantified[32] statement: $\forall x.\ bird(x) \rightarrow mammal(x)$.

5.4.1.1 Inheritance and Cancellation

One of the main advantages cited for semantic networks as a knowledge representation is the ability to "see" clusters of concepts. Within these clusters, the number of links between nodes represents a kind of semantic closeness. Given that general shared concepts are pushed up a taxonomy or hierarchy, those that remain close are less general and therefore share more common points.

The move to have concepts represented just once at the most general level necessitates having some mechanism to reclaim those properties. This is provided by inheritance in one of its many forms: if A is a subclass of B and B has property C, then A has property C. In general, (see below for exceptions) IS-A is a (!) transitive relation and the aim of a network is to compactly represent those concepts that IS-A other concepts.

Using IS-A links to represent subset/superset relations, the properties applying to individuals or classes could be found in a variety of means: a program could walk the network, traversing the IS-A links to find additional properties or concepts; makers could be propagated up the network and the properties so marked would apply; or a subnet could be constructed and attempts made to match it to the existing network structure.

Semantic networks as a collection of nodes with labelled links have no inherent logical structure. There is nothing preventing a network containing concepts "flying animal" and

[31] It should be noted that if leaf nodes are to be considered as tokens or individual objects (rather than individual classes that have a single member or predicates that hold for a single element in the universe of discourse [Brachman 1979]) then the IS-A link must be considered as a set membership relation or predication for leaf/non-leaf link [Brachman 1983].

[32] This ignores for the moment the possibly conditional nature of the IS-A link that arises when considering exceptions.

"non-flying animal" and having IS-A links pointing from another concept, "Tweety", to both (possibly though intermediate nodes such as "bird" and "animal" respectively). Thus a network can say without any idea of impending contradiction that Tweety both flies and doesn't fly.

By adding a polarity to links it is possible to use a single node to represent a concept and its negation in the same place, and perform some kind of processing so that only one of the two relations is allowed to apply at a given node. Thus if "Tweety" IS-A "flying animal" and IS-NOT-A "flying animal" then a contradiction is avoided by allowing one or other of these links to have precedence.

How precedence between conflicting relations is decided poses problems for simple propagating algorithms. The simplest algorithm is to apply the first relation that is found, so that if IS-A is traversed before IS-NOT-A then Tweety could fly. This means relying on implementation-specific details to determine the properties of a network, and potentially "trivial" changes (such a changing the order in which links are added to a network) result in changes to the interpretation of the network.

A common algorithm used (e.g. FRL or NETL) in an attempt to circumvent this problem, relies on choosing the conflicting relation that has the shortest number of links back to the original concept. Given the following network:

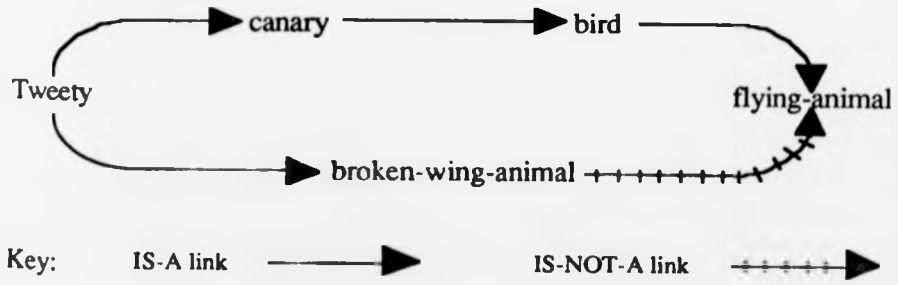


Figure 5.4.1.1 (a)

Tweety does not fly, as the shortest path uses a IS-NOT-A as its final link. But this algorithm

is also subject to "trivial" changes affecting the interpretation. Adding a redundant link, "Tweety" IS-A "bird" (which can be calculated from the network) there is no longer a shortest path and it is unclear (again) whether Tweety can fly or no.

Touretzky [1984, 1986] attempts a formal analysis of inheritance and presents an algorithm, the shortest inferential path algorithm, that is stable with regard to adding redundant information and can resolve some conflicts in inheritance. In the following sections I shall show: how the three different algorithms can be implemented using different interpretations of the same network; how the correspondence between some of Touretzky's theorems and definitions relate to IDN concepts; and finally show why Touretzky's algorithm canno t be implemented in the way he desires using parallel marker propagation machines.

All this demonstrates the usefulness of IDNs as a knowledge representation and analysis technique and shows how IDN concepts are common across yet another AI representation.

5.4.1.2 Interpretation of IDN-based Semantic Networks

The basic IDN reasoning technique is to construct complete interpretations of networks, corresponding to finding the closure of a set of axioms. There is less emphasis on relationships between particular nodes. This may not be the case with semantic network applications where the questions are of the type "Does x have property P".

IDNs still support this approach in a variety of ways. First, a partial admissible valuation could be created over the network connecting x and P. Second, some other type of interpretation mechanism could be used (e.g. dataflow processing). Finally, it may be possible to produce a definition of entailment based on admissible valuations and linked to the structure of the network, so that questions of the above type can be answered more easily.

None of these suggestions detract from the basic validity of using IDNs to capture semantic network information and processing. As long as the summation functions capture some approach to semantic network processing, then finding admissible valuations will be a correct way of deriving the appropriate information.

5.4.1.3 Outline of Semantic Networks as IDNs

The representation of semantic networks has three basic components: a set of dependency types to represent links found in the semantic network and to cover assertions (of facts); a set of values that are assigned to nodes to show which concepts are related; and a set of summation functions for the dependencies and nodes that govern how values are assigned and implement a particular inheritance algorithm.

IS-A (and IS-NOT-A) links are easily represented by a dependency with a single antecedent and consequent. However other relations (excluding IS-A and IS-NOT-A) will have a second antecedent that indicates the relational link that any particular dependency instantiates. This is done for the sake of flexibility and to reduce the number of types of dependency[33]. So the link "Tweety IS-A bird" becomes

$$\text{Tweety}_{\text{ant}} \longrightarrow_{\text{SN+}} \text{bird}$$

where Tweety and bird name unique nodes in the network[34]. "Tweety HATES cats" is represented by

$$\text{Tweety}_{\text{ant}}\,\text{hates}_{\text{rel}} \longrightarrow_{\text{SN+}} \text{cats}$$

and as before "Tweety", "hates" and "cats" are uniquely named nodes.

Negation or cancellation links are defined for both IS-A links and relation links, through the introduction of two more types of links:

$$\text{Tweety}_{\text{ant}} \longrightarrow_{\text{SN-}} \text{bird}$$

$$\text{Tweety}_{\text{ant}}\,\text{hates}_{\text{rel}} \longrightarrow_{\text{SN-}} \text{cats}$$

representing "Tweety is not a bird" and "Tweety does not hate birds" respectively[35].

---

[33] Without this facility each different relation, e.g. IS-A, "colour", "hates" etc, would have to be a different type of dependency with a different summation function. This approach ensures access to the interpretation of the network through assertional dependencies allowing a degree of flexibility in changing the network interpretation and structure in a dynamic fashion.

[34] In practice the nodes and links are represented using system-generated names and the external denotation is provided by a mapping between system names and user-defined names.

[35] Touretzky also introduces an unknown link "Tweety may be a bird or may not be a bird". However I am uncertain about the semantics he gives to this link and will therefore exclude this kind of link from consideration. The uncertainty arises from the fact that an unknown link cannot be overridden by positive or negative evidence - if a direct link saying Tweety has unknown status with regard to being a bird, then saying Tweety is a canary and canaries are birds, does not provide evidence for Tweety being a bird. However, if it is shown to

The only other kind of links are assertional dependencies that are used to introduce *tokens* into the network that are then propagated around the network to construct an interpretation. For example, $\longrightarrow_{v_1}$ Tweety identifies Tweety with the token v_1.

This then is the basic structure of semantic networks as IDNs and in the next section various algorithms for constructing interpretations will be given. In general this shall work as follows: a set of tokens will be defined; a way of interpreting assignments of tokens to nodes will be given; and a method of propagating and combining tokens that specifies a particular inheritance algorithm will be outlined.

5.4.2 Interpreting Semantic Networks

The simplest interpretation for semantic networks is to use a single token to represent a single concept c and propagate that token across IS-A links and relational links. If the token + arrives at node p, then this is interpreted as saying p holds for c, i.e. p(c) is true. In the course of its travels, + may be modified to become −, +r or −r where r is some other token. In this case its arrival at p is interpreted as follows:

−	≡	p(c) is false
+r	≡	r(c,p) is true
−r	≡	r(c,p) is false

The hybrid token +r is to be interpreted as a lambda expression $\lambda x.r(c,x)$ where x is either a concept name or instances of that concept.

The assignment of positive or negative tokens is done by SN+ or SN− dependencies that have no antecedents and therefore act as assertional dependencies.

The simplest inheritance algorithm is to propagate tokens in a non-deterministic, implementation-specific way - the first token to arrive at a node is the one that applies. Consider the following network:

---

be desirable then unknown links could easily be included in the definition of IDN semantic networks.

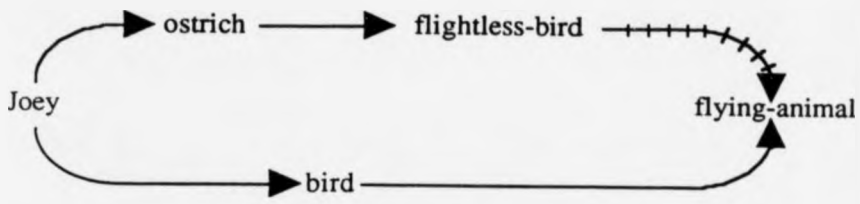


Figure 5.4.2 (a)

By assigning Joey a token + with an assertional dependency, one could construct an interpretation where ostrich was +, bird was +, flightless-bird was + and flying animal was + thereby preventing flying animal from being negative. Alternatively, values could be assigned to ostrich (+), flightless bird (+), and flying animal (−) even before bird (+) was considered.

The summation functions for this kind of interpretation are as follows, given v::w represents the concatenation of the two tokens v and w, and a set of tokens T is consistent iff $\{+,-\} \not\subseteq T$ and $\forall r \in \mathcal{R}, \{+r, -r\} \not\subseteq T$.

$$\mathcal{V} = Pw(\{ nil, +, - \} \cup \{ p::r \mid p \in \{ +, - \}, r \in \mathcal{R} \})$$

where $\mathcal{R}$ is some set of relation tokens

$$S^{N}(v_1, ..., v_m) = v \subseteq v_1 \cup ... \cup v_m$$

where v is a maximal and consistent but implementation-dependent

$$S^{D}_{SN+}(v_1, v_2) = \{+::r \mid v_1 = +, r \in v_2 \neq \varnothing\}$$

$$= \{v_1\} \text{ for } v_2 = \varnothing$$

[this assumes that non-labelled relational links are IS-A links]

$$= \{nil\} \text{ otherwise}$$

$$S^{D}_{SN-}(v_1, v_2) = \{-::r \mid v_1 = -, r \in v_2 \neq \varnothing\}$$

$$= \{v_1\} \text{ for } v_2 = \varnothing$$

[this assumes that non-labelled relational links are IS-NOT-A links]

$$= \{nil\} \text{ otherwise}$$

$S^{\wedge}_{SN+,a}(v_1, ..., v_m) = +$ if $\forall i. + \in v_i$ or $\{v_1, ..., v_m\} = \varnothing$

$= nil$ otherwise

$S^{\wedge}_{SN+,r}(v) = \{ r \mid r \in v, r \in \mathcal{R} \}$

$S^{\wedge}_{SN-,a}(v_1, ..., v_m) = -$ if $\forall i. - \in v_i$ or $\{v_1, ..., v_m\} = \varnothing$

$= nil$ otherwise

$S^{\wedge}_{SN-,r}(v) = \{ r \mid r \in v, r \in \mathcal{R} \}$

It should be noted that hybrid tokens representing lambda expressions do not propagate over any links once they have been created. It would be possible to create, as a straight forward extension, a type of link which does look for and propagate relational tokens. Given time and space considerations this has not been attempted. However, the simple extension to allow links with multiple antecedents has been included. This allows for the coding of simple conjunctive links: e.g.

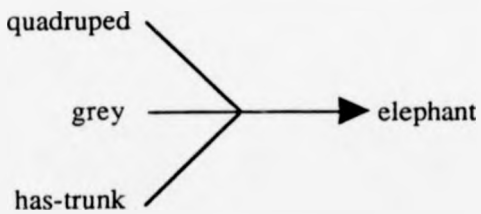


Figure 5.4.2 (b)

5.4.2.1 Shortest Path Algorithm

Keeping with the same basic representation (i.e. not introducing named tokens for concepts) a shortest path algorithm can be implemented by extending the set of values assigned to nodes:

$\mathcal{V}$ = values′ × Z where values′ are the values defined above

and Z is the set of non-negative integers

$S^{N}(v_1, ..., v_m)$

$= nil$ if $\forall i.v_i = \{nil\}$

$= \{<t_i, n_i> \mid t_i = p::r$ where $p \in \{+, -\}, r \in \mathcal{R}$ and $\forall j \neq i.\ t_j = q.r \rightarrow i < j\}$

$\cup \{<t_i, n_i> \mid t_i = p::r$ for some (implementation-dependent) choice of p from $\{+, -\}$

where $<+r, n_i>, <-r, n_i> \in v_1 \cup ... \cup v_m$

and $\nexists$ $<t_k, n_k> \in v_1.t_k = q::r \wedge n_k < n_i\}$

$S^{D}_{SN+}(v_1, v_2)$

$= \{<+::r, n> \mid v_1 = <+, n>,\ r \in v_2 \neq \emptyset\}$

$= \{v_1\}$ for $v_2 = \emptyset$

[this assumes that non-labelled relational links are IS-A links]

$= \{nil\}$ otherwise

$S^{D}_{SN-}(v_1, v_2)$

$= \{<-::r, n> \mid v_1 = <-, n>, r \in v_2 \neq \emptyset\}$

$= \{v_1\}$ for $v_2 = \emptyset$

[this assumes that non-labelled relational links are IS-NOT-A links]

$= \{nil\}$ otherwise

$S^{A}_{SN+,a}(v_1, ..., v_m)$

$= <+, n+1>$ if $\exists i.<+, n> \in v_i$ and $\forall j.<+, n_j> \in v_j \wedge n_j \leq n$

$= <+, 0>$ if $\{v_1, ..., v_m\} = \emptyset$

$= nil$ otherwise

$S^{A}_{SN+,r}(v) = \{\ r \mid r \in v, r \in \mathcal{R}\ \}$

$S^{A}_{SN-,a}(v_1, ..., v_m)$

$= <-, n+1>$ if $\exists i.<+, n> \in v_i$ and $\forall j.<-, n_j> \in v_j \wedge n_j \leq n$

$= <-, 0>$ if $\{v_1, ..., v_m\} = \emptyset$

$= nil$ otherwise

$$S^{\wedge}_{SN-,r}(v) = \{\, r \mid r \in v, r \in \mathcal{R} \,\}$$

Given this interpretation, the network above has a single admissible valuation V:

node	Joey	ostrich	flightless-bird	bird	flying-animal
value	<+, 0>	<+, 1>	<+, 2>	<+, 1>	<+, 2>

The value <−, 3> is not assigned to flying animal as the token <+, 2> inherits along a shorter path. However, the shortest path algorithm doesn't help in interpreting a network when there are no shortest paths. The network remains ambiguous with multiple interpretations[36].

5.4.2.2 Individual Tokens

Another dimension of complexity is to introduce named concept tokens rather than the simple {+, −} set of tokens. Instead, a set of *individual tokens* $I = \{i_1, i_2 \ldots\}$ is used to denote individuals, with hybrid tokens $+i_j$ and $-i_j$ used to indicate that a predicate applies positively or negatively to the jth individual, while +ri and −ri assigned to a node p indicates that r(i,p) is respectively true or false.

The summation functions defined above have to be quantified for each individual token so that

$$S^{\wedge}_{SN+,a}(v_1, \ldots, v_m)$$
$$= <+, n+1> \text{ if } \exists i.<+, n> \in v_i \text{ and } \forall j.<+, n_j> \in v_j \wedge n_j \leq n$$

becomes

$$S^{\wedge}_{SN+,a}(v_1, \ldots, v_m)$$
$$= \{\, <+i, n+1> \mid \exists i.<+i, n> \in v_i \text{ and } \forall j.<+i, n_j> \in v_j \rightarrow n_j \leq n \,\}$$

In order to account for the increased number of tokens, new assertional dependencies must be added. This is done through dependencies with no antecedents but that are typed according to the values they assign. The dependency $\longrightarrow_i p$ having a summation function

[36] depending on whether a particular interpretation allows none or many interpretations of an ambiguous network

$S_i^D(\varnothing)=i$

contributes the token i to p's node level summation function, S_{SN}^N.

5.4.2.3 Redundancy and the Inferential Distance Algorithm

Shortest path algorithms (SPAs) can solve some of the difficulties in interpreting networks: a partial order can be imposed on concepts to arbitrate between certain conflicting concepts. What an SPA cannot do is to take account of redundant information. Nor can networks be compounded in expected ways. Consider Joey the ostrich again:

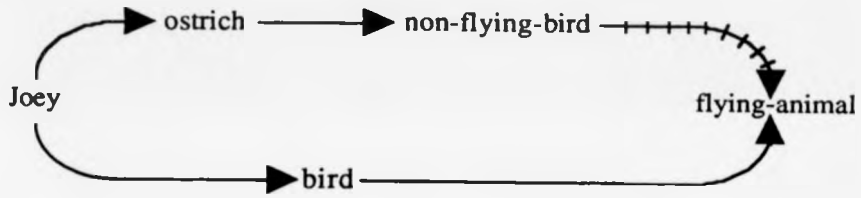


Figure 5.4.2.3 (a)

As the network currently stands Joey is a flying-animal using a SPA. Introducing what appears to be redundant information, that Joey IS-A non-flying-bird - something that the system calculates anyway, introduces ambiguity. There is now conflicting and equal evidence that Joey may or may not be a flying-animal.

A more interesting property of using an SPA is the non-compositionality of networks. Consider adding the link non-flying-bird IS-A bird. Looking at the network from this point upward it appears unambiguous in the light of SPA: non-flying birds are not flying-animals and cannot even appear to be without directly introducing contradiction into the network (e.g. by adding non-flying-animal IS-A flying-animal).

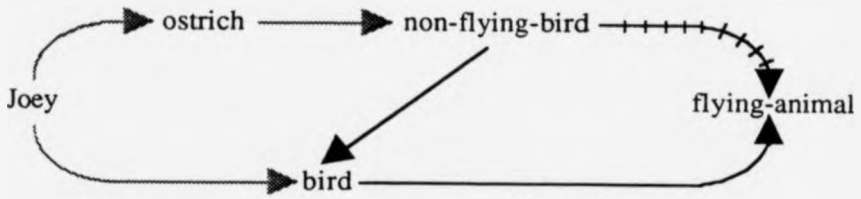


Figure 5.4.2.3 (b)

This network appears to say exactly what is required about non-flying-birds, birds and flying-animals. Yet adding information about Joey the ostrich (who is obviously a bird) distorts the intended meaning and there is a non-flying-bird who is also a flying-animal through virtue of being a bird. Obviously these problems could be circumvented through better structuring of concepts but that would miss the point: given an SPA, the intended meaning of a network can only be preserved by not allowing the addition of redundant or extra information.

Touretzky's Inferential Distance Algorithm prevents the addition of redundant information causing changes in the interpretation, by using the most specific paths in calculating interpretations. If there is both a single-step and a multiple-step relationship between two nodes then the single-step relationship is ignored. Competing paths are then compared using these maximal paths. In a sense this is equivalent to using some kind of longest path algorithm. When it comes to interpreting networks, single-step relationships win over multiple steps, while conflicting multiple-step relationships result in ambiguity, and conflicting single-step relationships result in contradiction or inconsistency. If there are two inheritance paths sharing a common initial subpath, followed by a single step in one case and a contradictory multiple-step relationship in the other, then the single step wins over multiple steps, while two conflicting multiple-step relationships result in ambiguity, and two conflicting single-step relationships result in contradiction or inconsistency. Because the addition of redundant information does not change the maximal path (as any redundant path short-cuts the maximal path) such changes to the network will not change its interpretation.

Other than solving some of the problems of SPAs, Touretzky's main contribution is in providing some formal analysis of the inferential distance algorithm. In particular he defines expansions (or variously, extensions) as consistent and closed sets of single and multiple set relations and proves the following "major theorems" [Touretzky 1984 p324]:

- every acyclic inheritance network has a constructable extension;
- every extension of a (finite) acyclic inheritance network is finite;
- an extension is inconsistent iff the network itself is inconsistent;
- the union of any two distinct extensions is inconsistent;
- a network is ambiguous (has multiple extensions) iff it has an *unstable*[37] extension;
- every extension of an ambiguous network is unstable

In the next few sections I will sketch Touretzky's definition of inheritance, show how it can be translated into summation functions for our previously defined network structure, show that expansions correspond to admissible valuations, show how the majority of results above are a natural product of IDN admissible interpretations, and finally explain why Touretzky found it impossible to get a correct distributed algorithm for constructing expansions.

5.4.3 The Mathematics of Inheritance Systems

The following definitions form the basis of Touretzky's algorithm and describe what should be produced by an IDA-based reasoner. For the sake of these definitions an inheritance network is a set of pairs <+x, p::y> where x and y are nodes in the network and p ∈ {+, -, #}. A sequence of nodes <x ,..., y> represents the chaining together of pairs to show the inheritance of relations or concepts. The pair <+Tweety, +bird> is read as Tweety IS-A bird while the sequence <+Tweety, +bird, +flying-animal> represents the fact that Tweety IS-A flying-animal through virtue of being a bird.

Defn T2.1[38]

The *consequences*, *cons*(Φ), of a set of sequences Φ

[37] this will be explained in the following section

[38] The theorems are numbered as they appear in [Touretzky 1986 p 39-44].

$= \{\langle x, y\rangle \mid \langle x, \ldots, y\rangle \in \Phi\}$.

Defn T2.2

A set of sequences Φ *contradicts* $\langle x_1, \ldots, x_n\rangle$ iff $\langle x_1, x'_i\rangle \in cons(\Phi)$ where x_i, x'_i have the same token or literal but a different sign - e.g. x = +bird and x' = -bird or #bird.

Defn T2.3

A token y is an *intermediary* to $\langle x_1, \ldots, x_n\rangle$ in Φ iff $\exists i.y = x_i$ or

$\exists\langle x_1, \ldots, x_i, y_1, \ldots, y_m, x_{i+1}\rangle \in \Phi$, $1 \le j \le m$, $1 \le i < n$. $y = y_j$. [i.e. there is a path from x_1 to x_{i+1} that goes through y courtesy of a subsequence between x_1 and x_{i+1}.]

Defn T2.4

A set of sequences Φ *precludes* $\langle x_1, \ldots, x_n\rangle$ iff $\exists\langle y, x'_n\rangle \in \Phi$. y is an intermediary to $\langle x_1, \ldots, x_n\rangle$ in Φ. [I.e. Φ precludes the multiple-step relation $\langle x_1, \ldots, x_n\rangle$ iff there is a single-step relation on the longest possible path from x_1 to x_n that would assert the opposite conclusion.]

Defn T2.5

$\langle x_1, \ldots, x_n\rangle$ is *inheritable* in Φ iff n>2 and $\langle x_1, \ldots, x_{n-1}\rangle$, $\langle x_2, \ldots, x_n\rangle \in \Phi$ and Φ neither contradicts nor precludes $\langle x_1, \ldots, x_n\rangle$.

Defn T2.6

A set of sequences Φ is *closed* under inheritance iff

$\forall s.$ s is inheritable in $\Phi \Rightarrow s \in \Phi$.

Defn T2.7

A set of sequences Φ is an *expansion* of S iff $S \subseteq \Phi$ and Φ is closed under inheritance.

Defn T2.8

A set of sequences Φ is *grounded* in S iff $\forall s \in \Phi$. s is inheritable in Φ.

Defn T2.9

A set of sequences Φ is a *grounded expansion* of S iff Φ is an expansion of S and Φ is grounded in S.

Defn T2.15

A set of sequences Φ is consistent iff Φ contradicts none of its elements.

Defn T2.16

A set of pairs Γ (i.e. an inheritance network) is *ambiguous* iff it has more than one grounded expansion.

Defn T2.17

An expansion Φ is *unstable* iff it contains sequences $\langle x, y_1, \ldots, y_n \rangle \langle y_1, \ldots, y_n, w \rangle$ $\langle x, z_1, \ldots, z_m \rangle \langle z_1, \ldots, z_m, w' \rangle$ and Φ precludes neither $\langle x, y_1, \ldots, y_n, w \rangle$ nor $\langle x, z_1, \ldots, z_m, w' \rangle$. [I.e. both $\langle x, y_1, \ldots, y_n, w \rangle$ and $\langle x, z_1, \ldots, z_m, w' \rangle$ would be inheritable in Φ unless contradicted by Φ. As $x, y_1, \ldots, y_n, w \rangle$ and $\langle x, z_1, \ldots, z_m, w' \rangle$ are mutually contradictory, at most one of them can be inherited at a time.]

The first nine definitions are used to describe what an IDA should produce, i.e. a grounded expansion of an inheritance network (or certain parts thereof). The last three definitions are used to describe under what conditions a network is contradictory, ambiguous or unstable, i.e. conditions under which the network doesn't have a single consistent interpretation. I shall now show how Touretzky's notion of inheritance can be reproduced in an IDN-based semantic network.

Touretzky's work is founded on the idea of grounded expansions: that is, sets of inherited relations that are closed and grounded. Closure ensures that all possible inheritable relations are included in an expansion while groundedness ensures only those relations that are inheritable are included. Imagine a valuation of a IDN-based semantic network (SN-IDN) where each concept node is labelled with the relations inherited by that node. If a set of summation functions can be constructed that capture the definition of the IDA then expansion will correspond to admissible valuations.

The correspondence of grounded expansions with admissible valuations is based on two conditions that follow from the definition of admissible valuations:

$$S^{N}(n, V) = V(n) \Leftrightarrow S^{N}(n, V) \subseteq V(n) \text{ and } V(n) \subseteq S^{N}(n, V)$$

Given $S^{N}(n, V)$ calculates all the paths inheritable at n, given the paths inheritable at the antecedents of n, the first condition requires that an admissible valuation includes all the

inheritable paths for all nodes, while the second condition requires that the admissible valuation contains only the inheritable paths.

Assuming deterministic summation functions we have proved that acyclic networks have a single admissible valuation. This means in the context of an SN-IDN that finite acyclic networks have a finite, constructible valuation.

The flaw in this argument is that summation functions for an SN-IDN cannot be deterministic if they are to exhibit ambiguous behaviour, i.e. have multiple admissible valuations. In order to have multiple admissible valuations the summation functions must be non-deterministic.

There are two ways to circumvent this problem. The first is to make the summation functions deterministic by maintaining all possible valuations simultaneously. The second is to develop a notion of non-deterministic IDNs and show that the acyclic networks of this kind still retain the properties of interest.

The following section will explore non-deterministic functions and prove general versions of the majority of Touretzky's theorems, i.e. theorems that hold for any IDN satisfying the necessary (abstract) pre-requisites on the structure of a network and its summation functions.

The section after that will briefly explore multiple-context network reasoners as an alternative way to dealing with ambiguity in network interpretations.

5.4.3.1 Non-Deterministic IDNs

In order to develop a theory of non-deterministic IDNs it is necessary to define what is meant by such a thing. The basic notion to be captured is that of a function that, for any given value in its domain, could return one of a number of possible values in its range[39]. Which value is to be returned is decided purely at random.

[39] In this respect the function is not a proper function (which is defined as a many- or one-to-one mapping), but I shall continue to call it this for the sake of continuity. Properly it should be called a relation.

Defn[40] A function f:D⟶R is *non-deterministic*, of *order* m, iff ∀d∈ D. ∃R′⊆R. f(d) ∈ R′ and $1 \leq |R'| \leq m$ and ∄n<m. f is non-deterministic of degree n.

The set R′ will be called the *sub-range* of f at d, written $R|_d$

If the sub-range were empty for some d∈ D then f would be undefined for some values. If the sub-range always contains a single element, i.e. f is of order 1, then f is an ordinary deterministic function.

Rather than try to identify the sources of indeterminacy in the summation functions of a network, the value of each node in a given network can be considered the result of a single function, taking a (partial) valuation of the network and returning a single value out of a choice of m elements, i.e. $V^{\mathbf{N}}$(n, V) is of order m. The sub-range of $V^{\mathbf{N}}$ at n given valuation V, will be a subset of the values $\mathcal{V}$, written $\mathcal{V}|_{n,V}$[41]. It should be noted that the sub-range is self-referential for cyclic networks. It may not possible to know what set of values a node might take without actually knowing the value the node will take! This is an area that has not been explored as only acyclic non-deterministic networks have been considered up to this point.

Where previously valuations were admissible if the valuation assigned the same value as that calculated by the summation functions, a valuation of a non-deterministic IDN is admissible if the valuation returns a value that **could** have been returned by the summation functions, i.e. is in the sub-range of $V^{\mathbf{N}}$ for n given valuation V.

Defn For a network **<N,D>**, V is an admissible valuation iff ∀n∈ **N**. V(n) ∈ $\mathcal{V}|_{n,V}$.

A function f:D⟶R is *deterministic*, of *order* m, iff ∀d∈ D. ∃R′⊆R. f(d) ∈ R′ and $1 \leq |R'| \leq m$ and ∄n<m. f is non-deterministic of degree n.

The following theorem uses this definition to show that finitely deterministic acyclic networks, i.e. networks which at every node return one of a limited (finite) set of values, have a finite set of admissible valuations. The non-determinism licences an implementation

[40] This is the obvious definition of a non-deterministic function e.g. [J. Gersting, 1982, "Mathematical Structures for Computer Science", Freeman and Co, San Francisco p75].

[41] Given the value of $V^{\mathbf{N}}$(n, V) is dependent on a particular network, being calculated from the dependencies attached to n, the sub-range should have some indication as to which network this applies to. However, for the sake of brevity this reference will be omitted.

to return one of a set of possible valuations yet still restricts what is deemed the "right" answer.

This can be achieved due to one basic fact - because an order can be established on the nodes in the (acyclic) network and only a finite set of valuations is allowed for the first n-1 nodes, and the choices for the nth is determined by those values, then only a finite set of values is admissible for the nth.

Thm If a network $\langle \mathbf{N},\mathbf{D} \rangle$ is acyclic and non-deterministic of order $m \geq 1$, then $1 \leq |VA(\langle \mathbf{N},\mathbf{D} \rangle)| \leq m^{|\mathbf{N}|}$

Proof By induction on $|\mathbf{N}|$

Because $\langle \mathbf{N},\mathbf{D} \rangle$ is acyclic, it can be decomposed into a series of subnetworks $\langle \mathbf{N}_i,\mathbf{D}_i \rangle$ for $1 \leq i \leq |\mathbf{N}|$ where $\mathbf{N}_{i+1} = \mathbf{N}_i \cup \{n_{i+1}\}$, $\mathbf{D}_{i+1} = \mathbf{D}_i \cup \{d_{i+1}\}$, and $cons(d_{i+1}) = n_{i+1}$ and $cons(d_{i+1}) \subseteq \mathbf{N}_i$. $\mathbf{N}_0 = \mathbf{D}_0 = \varnothing$.

Inductive Hypothesis

For any i and any m assume $1 \leq |VA(\langle \mathbf{N}_i,\mathbf{D}_i \rangle)| \leq m^{|\mathbf{N}_i|}$. As $D(n_{i+1}) = \{d_{i+1}\}$ and d_i contains no reference to n_{i+1} or its descendents, $V^{\mathbf{N}}(n_{i+1},V)$ is wholly determined by $V|_{\mathbf{N}_i}$ for any $V \in \mathbf{V}^{|\mathbf{N}|}$.

Furthermore, as $V^{\mathbf{N}}$ is non-deterministic of order m $V^{\mathbf{N}}(n_{i+1}, V) \in \{v_1, \ldots, v_p\}$ for $1 \leq p \leq m$ and any V.

So $\forall V \in VA(\langle \mathbf{N}_i,\mathbf{D}_i \rangle).\forall v_j.\ V \cup \{(n_{i+1}, v_j)\} \in VA(\langle \mathbf{N}_{i+1},\mathbf{D}_{i+1} \rangle)$ and $|VA(\langle \mathbf{N}_{i+1},\mathbf{D}_{i+1} \rangle)| = |VA(\langle \mathbf{N}_i,\mathbf{D}_i \rangle)| \times p$.

As $1 \leq |VA(\langle \mathbf{N}_i,\mathbf{D}_i \rangle)| \leq m$ and $1 \leq p \leq m$,

$1 \leq |VA(\langle \mathbf{N}_{i+1},\mathbf{D}_{i+1} \rangle)| \leq m^{|\mathbf{N}_i|} \times m = m^{|\mathbf{N}_{i+1}|}$

Base Case

For any m, $\mathbf{N}_1 = \{n_1\}$, $\mathbf{D}_1 = \{d_1\}$, $ants(n_1) = \varnothing$ and

$\exists V \subseteq \mathcal{V}.\ \forall V \in \mathbf{V}^{|\mathbf{N}|}.\ V^{\mathbf{N}}(n_1, V) \in V$ where $V = \{v_1, \ldots, v_m\}$.

Therefore $1 \leq |VA(\langle \mathbf{N}_1, \mathbf{D}_1 \rangle)| \leq m$ and by induction $1 \leq |VA(\langle \mathbf{N},\mathbf{D} \rangle)| \leq m^{|\mathbf{N}|}$.

The proof does not depend on m and $|\mathbf{N}|$ having a particular value and as such the proof holds for any order of non-determinism and size of network greater than or

equal to 1.

□

Note The theorem that every deterministic network has a single valuation is a special case of this, more general, theorem.

This theorem proves that every acyclic network, be it deterministic or non-deterministic, has one or more admissible valuations. As the proofs are constructive in nature (i.e. do not rely on a contradiction) it also proves that each network has one or more constructible admissible valuations. This reproduces Touretzky's Thm2.11. Furthermore, given: the network is finite; the proof shows that a value can be constructed for each node; and that the value is finite, the resulting admissible valuations must als o be finite (Touretzky's Thm2.8).

Touretzky's other main theorems deal with ambiguity and stability. To examine these concepts as properties of IDNs we first need to define them. The first is a straight forward translation of Touretzky's definition, and intuitively obvious.

Defn An IDN is ambiguous iff it has more than one admissible valuation.

Stability, as defined by Touretzky, is a very narrow concept tied into his network representation and Inferential Distance Algorithm. The basic idea is that an interpretation or expansion of a network is unstable if at any one point, either one of a pair of conflicting paths could be inherited. This is obviously related to non-determinism and gives rise to the more general definition for IDNs:

Defn Given a network **<N,D>**, an admissible valuation V is unstable iff $\exists n \in \mathbf{N}.\ |\mathcal{V}|_{n,V}| \geq 2$. I.e. there is a node s.t. the sub-range of $V^{\mathbf{N}}$ at n given V contains two or more elements, or again, $V^{\mathbf{N}}(n, V)$ is non-deterministic.

The following theorem is then a straight forward application of this definition. In brief, if a network is acyclic then the nodes can be ordered, and if there are two admissible valuations, there must be a first node at which the valuations disagree. But if all the ancestors have the same values in the two different interpretations, this can only happen if the valuation function at that point is non-deterministic.

Thm (Touretzky 2.13)

If an acyclic network **<N,D>** is ambiguous then each of its admissible valuations is unstable.

Proof As **<N,D>** is ambiguous, $\exists V_1, V_2 \in VA(\langle \mathbf{N},\mathbf{D}\rangle)$. $V_1 \neq V_2$. As $V_1 \neq V_2$ and **<N,D>** is acyclic $\exists n$. $V_1(n) \neq V_2(n)$ (*) and $\forall m \in ancs(n)$ $V_1(m) = V_2(m)$. If $\mathcal{V}^{\mathbf{N}}$ is deterministic then by the Isomorphic Parent Lemma, $V_1(n) = V_2(n)$, contradicting (*). Therefore $\mathcal{V}^{\mathbf{N}}$ is non-deterministic with order at least 2 and the valuations V_1, V_2 are unstable. The same argument applies to any pair $V,W \in VA(\langle \mathbf{N},\mathbf{D}\rangle)$ so all the admissible valuations of **<N,D>** are unstable.

The converse can also be easily proved. If a network has an unstable admissible valuation then the network is ambiguous, for a valuation is unstable iff two or more different values are admissible at a single node. Each of these values will give rise to a different admissible valuation and the network is therefore ambiguous.

Thm (Touretzky 2.14)

If an acyclic network **<N,D>** has an unstable admissible valuation then it has more than one admissible valuation.

Proof As usual, as **<N,D>** is acyclic an order can be defined on **N**. Given $V_1 \in VA(\langle \mathbf{N},\mathbf{D}\rangle)$ is unstable and that $n \in \mathbf{N}$ is the first node where $\mathcal{V}^{\mathbf{N}}$ is non-deterministic, i.e. $|\mathcal{V}|_{n,V}| \geq 2$, $\exists v \in \mathcal{V}|_{n,V}$. $v \neq V_1(n)$. As **<N,D>** is acyclic, values can be calculated for all nodes $\mathbf{N}'$ following n using the partial interpretation:

$V_2(m) = V_1(m)$ for $m \in \mathbf{N} \setminus [\mathbf{N}' \cup \{n\}]$

$V_2(m) = v$ for $m = n$

There is guaranteed to be at least one admissible valuation V' of **<N,D>** subsuming V_2 (i.e. V' and V_2 agree over $\mathbf{N} \setminus \mathbf{N}'$), otherwise $\mathcal{V}^{\mathbf{N}}$ would be undefined for some $n \in \mathbf{N}'$. As $V'(n) = V_2(n) \neq V_1(n)$, V' and V_1 are distinct admissible valuations of **<N,D>**, making **<N,D>** ambiguous.

5.4.3.2 IDN Coding of the Inferential Distance Algorithm

All the previous discussion of reproducing Touretzky's work given general results for non-deterministic IDNs relies on the existence of a set of summation functions capable of implementing the Inferential Distance Algorithm. In this section such a set of functions will be produced and shown to be correct.

The values assigned to nodes will be sets of sequences, where each sequence is made up of signed tokens. Each token represents a type or individual and the sequence denotes a transitive IS-A relationship between all elements of the sequence.

$\mathcal{V} = Pw(T_\bullet)$ where $T = \{ p::r \mid p \in \{ +, - \}, r \in \mathcal{T} \}$

and $\mathcal{T}$ is some set of individual and type tokens.

The summation functions can then be defined for the dependencies outlined in §6.4.3. The relational antecedent is used to build the inheritance sequences by indicating where the sequences are being inherited. Each dependency takes the sequences to be inherited and appends the new element to each sequence. At the node level, sequences that are contradicted or have intermediary sequences are removed, leaving a set of consistent and maximal (in the sense of being the longest or most complete) sequences.

For example,

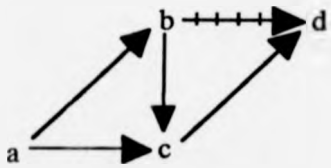


Figure 5.4.3.2

the sequences <+a +c> and <+a +b +c> are both propagated to c, but <+a +b +c> is an intermediary to <+a +c> so <+a +c> is removed. The sequences <+a +b +c +d> and <+a +b -d> are then both propagated to d, but as <+a +b -d> contradicts <+a +b +c +d>, <+a +b +c +d> is removed.

$S^{N}(v_1 \ldots v_m) = S \in Pw(T_*)$ where

$s \in S$ iff $\exists i.s \in v_i$ and

$\forall j \forall t \in v_j.t$ is not an intermediary to s and

$\forall u \in S.u$ does not contradict s.

Note: if $\exists s_i \in v_i$ and $s_j \in v_j$ s.t. s_1 and s_2 are contradictory but neither has an intermediary (i.e. both are inheritable) then one and only one of s_1 or s_2 must be in S. Which is to be included is the source of non-determinism - either is a valid choice.

$S^{D}_{SN+}(v_1, v_2)$

$= \{s * r \mid s \in v_1, r \in v_2 \neq \varnothing\}$

$= \varnothing$ for $v_2 = \varnothing$

where * is a function for joining two sequences

$S^{D}_{SN-}(v_1, v_2)$

$= \{s * r \mid s \in v_1, r \in v_2 \neq \varnothing\}$

$= \varnothing$ for $v_2 = \varnothing$

$S^{A}_{SN+,a}(v) = S^{A}_{SN+,a}(v) = v$

$S^{A}_{SN+,r}(v)$

$= \{ t \mid t \in v, t = +::r, r \in \mathcal{T} \}$

$S^{A}_{SN-,r}(v)$

$= \{ t \mid t \in v, t = -::r, r \in \mathcal{T} \}$

All that remains to be proved is that this set of summation functions produces the set of inheritable sequences dictated by Touretzky's definition (T2.5). This is done by showing that if maximal paths are propagated through a network, and if such a sequence is not contradicted or becomes non-maximal, then it can be inherited to the next node in a network. This is done using Touretzky's notion for clarity and ease of use for the appropriate definitions.

Thm If $s = \langle x_1, \ldots, x_{n-1} \rangle$ is inheritable and $\langle x_{n-1}, x_n \rangle \in \Gamma$ (where Γ is an inheritance network), and

there are no intermediaries to s, i.e. $\nexists \langle x_1, \ldots, x_i, y_1, \ldots, y_m, x_{i+1}, \ldots, x_{n-1} \rangle$, then $\langle x_1, \ldots, x_n \rangle$ is inheritable iff $\langle x_1, \ldots, x_n \rangle$ has no intermediaries and is contradicted.

Proof For $\langle x_1, \ldots, x_n \rangle$ to be inheritable $\langle x_1, \ldots, x_{n-1} \rangle$ and $\langle x_2, \ldots, x_n \rangle$ must both be inheritable and $\langle x_1, \ldots, x_n \rangle$ must be neither contradicted nor precluded. It can be shown that if $\langle x_1, \ldots, x_{n-1} \rangle$ has no off-path intermediaries then $\langle x_2, \ldots, x_n \rangle$ is inheritable iff $\langle x_1, \ldots, x_n \rangle$ is not contradicted and so the theorem holds.

All that is required then is to show that $\langle x_2, \ldots, x_n \rangle$ is inheritable.

As $\langle x_1, \ldots, x_{n-1} \rangle$ is inheritable, $\langle x_i, \ldots, x_{n-1} \rangle$ is inheritable for all $1<i<n-1$. So if $\langle x_j, \ldots, x_n \rangle$ were inheritable for some $2<j<n$ and $\langle x_{j-1}, \ldots, x_n \rangle$ were neither precluded nor contradicted then $\langle x_{j-1}, \ldots, x_n \rangle$ would be inheritable (as $\langle x_{j-1}, \ldots, x_{n-1} \rangle$ is inheritable).

So if $\forall 2<j<n. \langle x_{j-1}, \ldots, x_n \rangle$ is neither contradicted nor precluded then $\langle x_1, \ldots, x_n \rangle$ is inheritable.

If there are no off-path intermediaries to $\langle x_1, \ldots, x_n \rangle$, $\langle x_j, \ldots, x_n \rangle$ is precluded iff $\langle x_j, x'_n \rangle \in \Gamma$ for $i \leq j$. If this were the case then $\langle x_1, \ldots, x_n \rangle$ would be contradicted (contradicting the premises of the theorem).

Similarly, $\langle x_j, \ldots, x_n \rangle$ is contradicted iff $\exists \langle x_j, z_1, \ldots, z_p, x_n \rangle$ but if $\langle x_1, \ldots, x_n \rangle$ has no off-path intermediaries this is only possible if $\langle x_j, x'_n \rangle \in \Gamma$ where $x_j = z_k$ so $\langle x_j, \ldots, x_n \rangle$ cannot be contradicted without contradicting $\langle x_1, \ldots, x_n \rangle$.

So if the longest (maximal) sequences are propagated so that no off-path intermediaries exist, then a sequence can be inherited iff it is not contradicted or becomes non-maximal.

□

This proves that the summation functions outlined above are a correct implementation of the IDA.

5.4.4 Multiple Context SN-IDN

An alternative to having non-deterministic summation functions would be to maintain all possible interpretations through the use of a multiple context reasoner. This is an obvious way of avoiding non-determinism - rather than having a function return one of a set of possible values, have the function return the whole set.

In the case of SN-IDN this would mean assigning each node a set of sets of sequences where each set of sequences are contradictory. Consider the network:

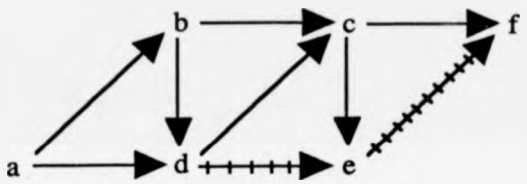


Figure 5.4.4 (a)

If the token a were propagated around the network, with tokens b,c,d,e,f used to label dependencies attached to B, C, D, E, and F respectively then the following admissible valuation would result:

Node	Value
A	{{(+a)}}
B	{{(+a +b)}}
C	{{(+a +b +c)}}
D	{{(+a +d)}}
E	{{(+a +b +e)} {(+a +d -e)}}
F	{{(+a +b +c +f)} {(+a +b +e -f)}}

Two sequences are inheritable at E: (+a +b +e) and (+a +d -e). As they are contradictory but neither is precluded they must be put into separate sets. Similarly, F has two possible sets of sequences it could inherit.

Having constructed a single admissible valuation, in order to work within a single context, it is necessary to choose a consistent set of sequences in the same way that a particular ATMS context must be constructed from a set of node labels. This is done by choosing an initial set of sequences and then including all sequences that are super-sequences of these, given the constraint that they must all come from the same subset of sequences assigned to a node. For example the sequences (+a), (+a +b), (+a +c), (+a +d) are all consistent super-sequences of (a). One of the subsets {(+a +b +e)} or {(+a +d -e)} must be chosen, and having done so, the choice of (+a +b +c +f) or (+a +b +e -f) is constrained by whether (+a +b +e) has been included in the current context - if it has, a free choice is possible, otherwise (+a +b +c +f) as (+a +b +e -f) is not supported.

The ambiguity and instability can be defined in terms of the number of contradictory sets of sequences and the conditions that apply to make two sets contradictory. The results that hold for non-deterministic IDNs should go through with the appropriately modified definitions.

The multiple context approach to dealing with ambiguity is less simple than the use of non-deterministic functions. It uses a more complex set of values and summation functions. It also requires a secondary interpretation to extract a single consistent context. However it does give the added advantages of being able to consider all possible contexts given just a single initial interpretation. It is also possible to reason about those contexts and choose one that contains a specific set of sequences. For example, if a context were required where -f was inheritable from a, it would require the sequence (+a +b +e -f) which would in turn require (+a +b +e) which would in turn exclude (+a +d -e).

5.4.5 Conclusion

In the preceding two sections, two approaches to dealing with ambiguity in inheritance networks have been outlined. Both rely on assigning complex values to nodes in the network. In the first case, sets of inheritable sequences are calculated while the second uses sets of sets of sequences.

In order to deal with redundancy through preclusion it is necessary to use such complex values, i.e. have sequences propagated through the network, rather than rely on single tokens. Without the information contained in the sequence that explains how a particular token was inherited it is impossible to decide, through a purely local computation given the information propagated to a single node, whether a given sequence is precluded or contradictory or inheritable. By limiting the local computation to single values (i.e. tokens), Touretzky's approach will never produce correct results, without conditioning.

This conclusion arises out of the study of the IDA and its re-implementation as an IDN. Other results of this work related to Touretzky's work are the general results produced in §5.4.3.1 on the constructability of interpretations and the origin of ambiguity in a network that apply to all non-deterministic IDNs.

The study of implementing (or viewing) semantic networks as IDNs has many benefits. In addition to showing how the framework of IDNs can be used as an analysis tool, it shows how one network structure is capable of supporting many different types of interpretation. It also provides a good example of dealing with contradictory or ambiguous interpretations: either resolve the contradiction and maintain a single interpretation (as with the JTMS) or move to a multiple context system and maintain all possible contexts (as with the ATMS).

Finally, it has shown how the IDN can be used to provide a formal tool for analysing the desired behaviour of network algorithms. By identifying semantic networks with IDNs and grounded expansions of the former with admissible interpretations of the latter, the desire for potentially ambiguous behaviour leads to a non-deterministic approach, which in turn leads to multiple possible interpretations thus providing the required ambiguity. However, in order to resolve apparent ambiguity in cases where it is not warranted, it is necessary to use more complex values in interpreting the network.

5.5 Summary

This chapter has shown how IDNs can be used to represent structures from many existing AI knowledge representation paradigms. This has not only shown the flexibility of IDNs as a representation tool but has provided a number of case studies that illustrate the following:

- IDNs give flexibility within a particular representation or structure to change the interpretation of that structure as shown by: the reuse of one basic network representation for Semantic Networks to implement different inheritance algorithms; the use of different uncertainty measures for Rule-Based Systems; and the development of a NATMS as derivative of the ATMS;
- IDNs provide an easy way of specifying network algorithms as shown by the preceding examples;
- IDNs provide a framework for the analysis of network-based algorithms: in particular highlighting the source of multiple extensions for Default Logic and generalising Touretzky's work on Semantic Networks and the ambiguity and constructibility of their interpretations;
- IDNs show, by splitting the representation into basic elements, that what is represented is just as important as how it is represented, as shown by the need to capture the right, "intuitive" inferences in Default Logic; and finally
- IDNs, through their approach to belief revision and continuous support, give rise to new developments of existing paradigms, as illustrated by defeasible inference within Rule-Based Systems.

In addition to showing the usefulness of IDNs as a theoretical and practical development, this chapter highlights several basic themes or ideas with regard to IDNs. First, self-referential support (i.e. circularities in networks) gives rise to many problems: they are computationally intractable; there is not always a guaranteed interpretation of such a network; and they are one source of ambiguity in networks. Secondly, two approaches to dealing with contradictions repeatedly arise: either work within a single context and attempt to change contexts as it becomes inconsistent; or construct all possible contexts using a multiple context reasoner. Despite the flood of literature supporting a multiple context approach as epitomised by the ATMS, single context systems have their own merits. By being able to operate in both modes through changing interpretation structures, IDNs can transcend this division.

CHAPTER 6

Conclusion

"And now, the end is near and so I face the final curtain ..."

Little now remains of this thesis, except to look back at what has gone before and to point the way forward to what yet may come. The research that led to this thesis has suffered from the problem of trying to keep the subject area within manageable bounds.

What started out as an in-depth review of Truth Maintenance Systems with a view to providing a generalised TMS led into ever expanding areas, in an attempt to understand on a technical and pragmatic level what it was that TMSs did, and were used for. This happened on two divergent fronts: first into non-monotonic logics and then general belief revision on a technical level; and secondly into rule-based systems, semantic networks, and problem solving and representational issues on a more pragmatic level. All this combined to produce the idea of an Interpreted Dependency Network.

In order to bring together the large amount of ground covered by this research, the following sections attempt to summarize the thesis and its contribution, and also highlight some of the areas that the thesis could have spread to cover but did not.

6.1 The Thesis in (less than) 100 Words

Belief revision is the process of adding and subtracting information to some model in the light of external events and changes in internal state through processes such as reasoning and maintaining consistency. In order to produce non-monotonic behaviour and maintain consistency in an environment in which external factors represented by premises, assumptions and/or defaults in the model can change, it is necessary for each proposition to have a set of explicit justifications for its current level of belief, and for the belief in a proposition to wax and wane as the support provided by the justifications changes. This is the general philosophy that lies behind existing Truth Maintenance Systems.

6.2 The Solution in (less than) 50 Words

Interpreted Dependency Networks provide: a way of specifying belief values and types of justifications; a definition of how propositions should be interpreted, i.e. assigned a belief value, given a particular set of justifications; and an algorithm for constructing exactly those interpretations. In particular, IDNs can be used to reconstruct existing Truth Maintenance Systems in a single unified structure.

6.3 The Contribution

The work in this thesis contributes in a variety of ways to the development of AI in general. It contributes:

- to work on TMSs through a thorough review and analysis of existing systems;
- to extending and generalising various existing AI results, representations and mechanisms through the application of IDNs to those domains; and
- to belief revision in general by demonstrating the benefits of a foundations approach and by showing the necessity of constructing a model that captures the right set of inferences by explicitly building them into the model.

In addition to defining IDNs and making the above contributions, the examples contained in the thesis provide a set of cases showing how IDNs can be applied and what benefits can result.

6.3.1 Truth Maintenance Systems

Chapter 2 provides a comprehensive review of existing styles of TMSs (§2.1) along with existing unifications of TMSs (§2.2). What emerges is a deconstruction of TMSs into three parts: a network of dependencies; an interpretation of the network based on labelling nodes; and the alteration of the network structure to perform such tasks as maintaining consistency and automatically extending a network.

Another result of the review is the finding that any generalisation of TMSs must be able to

- handle automatic updating of values through Truth Maintenance,
- represent and perform default inferences, and
- handle changing sets of assumptions

as these characteristics occur separately in existing systems.

The fact that IDNs are a successful generalisation of existing TMSs can be seen both by the reproduction of existing TMSs as IDNs (in §3.3) and the construction of an extended, non-monotonic ATMS (the NATMS defined in §5.1) that shares the same network structure as the JTMS and needs no extra machinery outside the network in order to construct admissible interpretations. Although an IDN is not necessary for the implementation of the NATMS labelling scheme and network structure, this section (§5.1) amply demonstrates how TMSs can be easily implemented as IDNs. To change from an ATMS to an NATMS it is only necessary to re-define the summation functions in two places.

A number of smaller contributions include providing a sound, complete and terminating algorithm for the JTMS, along with an analysis of the complexity of constructing interpretations of JTMS networks. Although it has been shown that this is a non-polynomial problem, Chapter 4 shows that the complexity is determined by the number of cutsets, and the arrangement of the cyclic and acyclic components of a network. The number of possible interpretations of a cyclic component is exponential in the number of possible values raised to the power of the number of cut-points, while the number of possible interpretations of a network is the product of the number of interpretations for interacting cyclic components. For example if component A depends on components B and C, then $|VA(A)| \leq |VA(B)| \times |VA(C)|$, and for a set of values $\mathcal{V}$ and cut-points CP_C for C, $|VA(C)| \leq |CP_C|^{|\mathcal{V}|}$.

6.3.2 Application Domains

In addition to the NATMS showing how IDNs provide a vehicle for generalising and synthesising different TMSs, other application areas are examined in Chapter 5 in order to demonstrate the use of IDNs and provide support for the thesis.

One key part of the thesis is that links between propositions, i.e. inferences, should be explicitly represented by dependencies. This is demonstrated in §5.2 when Morris' attempt to solve the problem of multiple interpretations is examined. He succeeds in solving the problem only by carefully choosing which inferences should be included in a particular network. My work shows that a more careful (but still intuitively appealing) encoding of the problem based on including contra-positive forms of implications results in the return of multiple interpretations. It is possible to produce the "right" (intuitive) results in default reasoning (using any particular system) but it requires the explicit representation of dependencies between different types of abnormality.

Propositional rule-based systems (PRBSs) are an excellent example of a type of system where links between propositions are represented explicitly. Given this, it should be no surprise that they are easily representable as IDNs. However, this is not the full story. Such systems usefully work on a "match-select-fire" iteration whereby rules are examined to see which antecedent conditions match with known facts, and one rule out of those that match is selected to be fired, with the consequent being added to the set of known facts.

Representing PRBSs as IDNs involves a new paradigm. Rather than matching, selecting and firing and then forgetting the rule, using IDNs means there is a persistence between antecedent condition and consequent result that is not normally present[1]. This gives rise to a defeasible (not necessarily non-monotonic) rule-based system. This means that principled non-monotonic rule-based systems can be constructed at little extra cost and without some of the problems that can result.

Finally, one of the benefits I claim for IDNs is that having a formal structure to the interpretation provides a means for analysing network-based algorithms that can be framed

[1] Parallels can be drawn between this model of interaction and that of spreadsheets.

as IDNs. This is demonstrated by showing how a variety of existing semantic network inheritance algorithms can be implemented as different interpretation mechanisms on top of a single unified network representation. A desire to produce ambiguous behaviour in such networks (in order to demonstrate how the Inferential Distance Algorithm works) leads to a non-deterministic paradigm for IDNs. The study of this type of network shows that most of Touretzky's work on proving the existence and stability of interpretations of networks using the IDA is a product of the non-determinism, rather than any other particular property of the IDA.

6.3.3 Belief Revision

The major contribution of this thesis is in supporting the foundations approach to belief revision. Little work, bar that on argument structures, has been done on this approach - most work has focussed on a consistency approach. This thesis has attempted to show that a foundations approach is not only preferable to a consistency approach (as argued in §1.2.3) but is feasible. In attempting to argue this I have thrown up some interesting problems arising from the consistency approach, as put forward by Gardenfors. One such result is that Gardenfors' postulates for belief revision are incompatible with a foundations approach - the postulates themselves embody a consistency approach, and in order to deviate from that approach, it is necessary to change the postulates.

Obviously the main result of the thesis must be the definition of the IDN for without this framework it is not possible to make all the other contributions mentioned above. It would be impossible to formalise the intuitive notion of "support" without some kind of formal mechanism such as that provided by the definition of dependencies and summation functions. Without the notion of an admissible valuation as a closed and consistent interpretation of a network, it would not be possible to show in detail how a foundations approach to belief revision could be applied to rule-based systems to get defeasible reasoning nor how multiple interpretations depend on cyclicity in a network or non-determinism in interpretation functions.

IDNs provide a test bed with which to explore the use and limits of a foundations approach to belief revision. Without it, the championing of a foundations approach is little more than speculation.

6.4 Future Applications

The contributions made in the thesis to various application domains provide a guide to potential applications of IDNs above and beyond those mentioned in Chapter 5 of the thesis. These fall into two broad categories: practical applications; and theoretical applications.

Practical applications are those that make use of IDNs in building representation systems and/or practical problem solvers based on a model of some domain. Any distributed network-based algorithm might be represented using IDNs, and the ease of specification through the definition of dependencies and corresponding summation functions means it is easy to produce tailor-made belief revision or Truth Maintenance Systems. This thesis provides a selection of such interpretation schemes along with several ways of tackling problems, i.e. using single or multiple contexts with or without quantitative measures of belief. A variety of approaches to reasoning with a model are also provided in the thesis along with a general purpose interpretation algorithm which can be tailored to suit particular problems.

On the theoretical side, IDNs provide a useful analysis tool. In order to represent a problem or model as an IDN, two things must be made explicit. It is necessary to deconstruct any interpretation mechanism into a set of (distributed) summation functions. This forces an analysis of the semantics of the interpretation, as for example happens with the inheritance algorithms in §5.4.

Secondly, by forcing the links between propositions to be explicitly represented by dependencies, it is necessary to examine the connections between propositions. In this way the interactions between data must be analysed and an otherwise intuitive appealing representation of a problem may be shown to have limitations caused by missing inferences (as shown in §5.1) or unwanted conclusions as the result of extra dependencies.

Having constructed an IDN it then becomes a simple matter to prove the termination or complexity of an algorithm based on the complexity of the summation functions and the general interpretation mechanism. Similarly, the topology of the network in connection with the properties of the summation functions can be used to determine whether it is ambiguous.

6.5 Further Work

In the course of this research two areas have been examined in passing but have not been followed up due to constraints on time. These have been in the investigation of *consumers* and the addition of learning algorithms for defeasible RBSs.

6.5.1 Consumers

The term "consumers" originates in work done by de Kleer [1986]. The basic idea is to have a procedure that is attached to a node which is run when a particular label is calculated for a node. The result of running this procedure is to make some change to the structure of the network by adding extra nodes, dependencies, constraints on the problem solving environment, or even other consumers. This provides a mechanism for extending the network in certain predetermined ways with dependencies or nodes which are not initially required in the network. This in turn provides a way of controlling the construction of interpretations for the network by incrementally constructing the network. It also provides a way of performing reasoning about the internal structure of nodes, i.e. the proposition represented by a particular node. So, for example, consumers could be used as a mechanism for pattern matching in an RBS-type system.

It would be interesting, and in some cases necessary, to have this type of mechanism in order to increase the flexibility and power of IDNs. One restriction on de Kleer's consumers is that the arguments passed to the consumer are limited to the antecedents of the justifications of the consumer's node. One possible extension to the idea of consumers would be to allow the use of network-traversing algorithms or other pre-defined functions. For example, an upward traversal might be used to trace the origin of support for a particular value by traversing only those dependencies that propagated the value in question. This is

precisely what is done by the first step of Doyle's [1979 p19] dependency directed backtracking algorithm.

Allowing consumers to be added to an IDN, be they as restricted as de Kleer's in terms of inputs and allowed outputs, or merely limited to a set of pre-defined functions, or more generally allowed to contain arbitrary procedures, has certain drawbacks.

A network interpretation may rely on particular properties of the network. It may be that only acyclic networks are allowed, or that acyclic and cyclic networks are interpreted differently, or that any cycles must be monotonic. If a consumer is allowed to add dependencies to the network there is no guarantee that these properties will be preserved.

Even if the network is interpretable given any set of possible actions there is great difficulty in predicting the consequences of allowing consumers. It is possible that a consumer will be fired given a certain value and the outcome of running the consumer will be the retraction of the value that led to the firing. There are no dependencies between consumer antecedents and conclusion, and if there were, this could lead to unsatisfiable cycles - consider a consumer which is fired but invalidates its own firing condition. The action would have to be undone, revalidating the consumer's antecedent condition and leading to a re-execution.

There is also the issue of the interaction between constructing interpretations and firing consumers. One approach is to construct an interpretation, fire a consumer to modify the network, and interpret the extended network before firing another consumer and so on. This is a conservative approach that means that many intermediate interpretations must be constructed before reaching a final interpretation. There would also be problems in scheduling the consumers. Different schedules could produce different sets of dependencies with different interpretations of the same set of nodes.

A more radical approach would be to fire all consumers that are valid after the first interpretation. This runs the risk of doing too much work by extending the network unnecessarily - if a conservative approach had been taken instead some antecedent conditions may have been invalidated and the consumers not fired.

In addition to consumers causing problems deciding exactly what interpretation of which network is the intended one, there is an even bigger problem in determining the complexity of such procedures. If it is not known what interpretation will be constructed it is even harder to decide how long that would take.

6.5.2 Learning Algorithms for RBSs

The defeasible rule-based system (DRBS) outlined in §5.3, when combined with an uncertainty measure such as certainty factors, bears a strong resemblance to a simple neural network. Both consist of directed graphs (although the DRBS allows for multiple antecedents or sources for edges or dependencies) and both label the nodes and edges from a set of numbers. In the same way as the value assigned to a node in an admissible valuation of an IDN is a function of the values of the antecedents, the value assigned to nodes in neural networks is a function of the strengths of the links. Finally, both the neural network and the DRBS can be viewed as a distributed representation of a function that maps input patterns to output patterns.

In the same way that neural networks adjust weights of arcs to minimise the error between the output of the network (given some input pattern) and the expected result (corresponding to the input pattern) in some set of training patterns, the DRBS could be tuned by adjusting the certainty factors associated with each rule. This opportunity is not available to normal RBS as the output is not merely a function of the inputs - it is also a function of the order in which rules fire.

Rather than have a neural network decide the strengths of connections (correlations) between inputs and outputs (often in an undecipherable way) by adjusting weights accordingly, trainable DRBS holds out the opportunity of a knowledge engineer specifying the causal connections between concepts, and letting a learning algorithm decide the relative import of the rules, by running it over a set of training cases. In this way, certainty factors become less of an intuitive judgement made by an expert and more of a meaningful correlation between data. The knowledge engineer may provide a starting point for the system, and in cases where there are two or more (local) minima in the error function that

determines the correlations, the starting point may determine which solution is chosen for the rule weighting. However, in this kind of system it would be the primary responsibility of the knowledge engineer to specify a basic causal model and it would be up to the learning system to tune that model so it best fits the available cases.

The success of this kind of approach would rest on the convergence of the error minimising process over a multi-layered network, such as back propagation [Rumelhart et al 1986]. Although I have conducted experiments on the convergence of hand-crafted functions extracted from a network structure, I have not attempted to do this on a large scale using automatically generated networks and automatically defined error functions.

6.6 Implementation

The focus of this work has not been on implementation but specification - it is more about what to do in a conceptual sense than about what to do in a programming sense. In this respect most of this section could be included in the further work section.

There are three basic parts needed for any implementation. The first is the data-structures needed to represent the network; the second is a set of low level functions or procedures needed to access and manipulate the structure of the network (e.g. the *cons* and *ants* functions) and to calculate the values of elements of the structure (i.e. the summation functions); and the third is an overall mechanism for controlling the creation of interpretations, providing functionality to edit the network and for running other procedures over the network.

Of these three levels, the first two have been implemented in LISP to a sufficient degree to allow for the construction of a simple interpretation mechanism for acyclic networks and also for a construction strategy based on simulated synchronous message passing. In this implementation, each node would repeatedly calculate its value until its value was admissible. Changes in a node's value would be propagated by notifying consequent nodes that they needed to (re)check for admissibility. This paradigm was abandoned as it was incapable of interpreting cyclic networks and was inefficient on a traditional single processor, von Neumann architecture. Nevertheless, this implementation was sufficient to act

as a test bed for the IDN summation functions (e.g. for certainty factors, ATMS and JTMS interpretations, Inferential Distance Algorithm) described in the thesis.

The algorithms included in Chapter 4 represent a number of procedures capable of constructing interpretations and are described in enough depth to be able to prove properties such as completeness and termination. They are also described at a sufficiently high level to be implemented in any number of ways, in any number of languages - there is little constraint in terms of machine architecture or programming constructs (beyond the ability to call sub-routines or procedures and to iterate). What the algorithms do constrain is the style of programming. There is an implicit procedural style of programming at a high level.

Despite this implicit constraint in this thesis, the typed network representation along with the distributed natures of the summation functions and the calculation of values for antecedent classes, dependencies and nodes, suggests that there might be benefits to be gained from employing an object-oriented approach based on message passing, or from using a parallel architecture.

Parallelism is often seen as a way of gaining computational benefits - having multiple processors means that algorithms can potentially run orders of magnitude faster than they would on conventional serial architectures. However these advantages can only be realised if the algorithm can be decomposed into suitable pieces to be distributed over the available processors, and if the overhead of communicating between processors is kept at a level below that of the benefits derived from the multiple processors. Another problem that restricts the benefits achievable is the need for synchronisation between processors. If one processor has to wait for another to finish before it (the first processor) can complete its task and it can do no other work in the meantime, the speed up will be less than if the synchronisation were not needed.

The task of calculating a node's value, given values for an antecedent is an easily identifiable piece of work and an obvious level at which to distribute the algorithm across a group of processors, assigning one or more node to a processor. Furthermore, given a node's value is a function of the values of its antecedents, the communication needs for a particular node are determined by the topology of the network and can be efficiently managed by

distribution of nodes to processors in such a way as to most closely match that topology. Despite these solutions to the problems of decomposition and communication, the problem of synchronisation is a limiting factor in IDNs. A node's value will be admissible only if calculated from antecedent values which do not subsequently change. This means that calculation must be postponed (or only partially completed) until all antecedent values have been calculated. This in turn means the maximum amount of parallelism is determined by the breadth of the network - i.e. how many nodes can be calculated at any one time, according to depth. Efficiency gains will be limited to some percentage of |N|, with the percentage gain determined by the topology of the network. Despite the relatively small size of gain (in terms of polynomial complexity), the speed-up could be significant (in absolute terms) for large and/or broad networks.

Object-oriented programming offers the benefits of easy specification, code reuse, and extendibility. The ease of specification comes from a data-oriented approach to design - it is easy to determine what objects are needed in the system (nodes and dependencies) and the procedures needed to interpret them (summation functions). The code reuse comes from two places - the sharing between objects of the same type in a system and sharing by defining similar objects in different systems. The extendibility is also a product of the data-oriented design - new functionality is added to a system by defining how classes of objects respond to new requests or methods.

The definition of IDNs exhibits many of the qualities needed for an object oriented approach. The definition of the network (§3.2.1) explicitly mentions different types of nodes, different types of dependencies attached to nodes and different types of antecedent classes within those dependencies. Each object's type determines its summation function and where the values needed as arguments for that function should be obtained, and these in turn define how an object is to be interpreted. By defining the summation functions, the designer specifies the interpretation mechanism (the origin of the argument values being defined by the valuation functions in §3.2.2 and the topology of the network).

Collections of objects such as cyclic or acyclic components can also be viewed as typed objects with different interpretation functions (cf §4.1 vs §4.2). Given the components of a

network themselves form a network, many of the same concepts (e.g. topological ideas of antecedents and consequents) apply. This is so much so that an interpretation algorithm for non-deterministic acyclic networks could be applied to the network of components by substituting a component valuation function for a node valuation function.

Code reuse across different IDN systems can be seen in §5.1. The NATMS borrows the network structure from the JTMS-IDN and borrows some of the summation functions from the ATMS-IDN. Similarly an uncertainty measure based on Dempster-Shafer theory might reuse ATMS-IDN functions.

Finally, having defined the network structures and summation functions in order to allow for the construction of admissible valuations, new interpretations (e.g. to check consistency as in §5.3.4 or use a different inheritance algorithm as in §5.4) could be constructed either by adding new methods to objects, or by redefining existing ones.

All the approaches, benefits and drawbacks described in this section (beyond the implementation covered in the first three paragraphs and the examples of object typing and code reuse) are hypothetical. Further work needs to be done in investigating typical network topologies, size, and the implications for the long term storage of information in IDNs in order to assess the potential for parallelism. It is also necessary to investigate how the construction of interpretations in an object-oriented system can be controlled via message passing in order to assess the practicality of such an approach. A research project is never finished!

APPENDIX

Counter Example to Goodwin's JTMS Algorithm

This appendix is included to show that the algorithm described by Goodwin [1987, p83] is in fact incorrect as given[1]. To see this consider the following network.

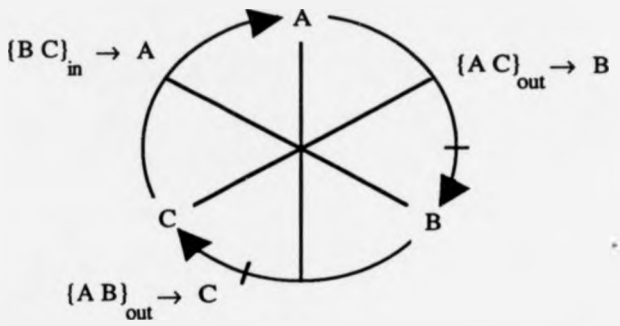


Figure A1 (a)

Imagine that all the nodes are unlabelled. This could happen for example by the removal of justification supporting both B and C. Alternatively the network could be a single strongly connected component in a larger network which is in the process of having its labelling recalculated.

Goodwin's algorithm goes through 3 steps (the step/function names below are taken directly from Goodwin's algorithm) while there are any unlabelled nodes in a strongly connected component S.

(1) Propagate.constraints (S)

(2) Find.not.well.founded (S)

(3) Partition (S, *in*, *out*)

Each step may return t, in which case the latter steps are not executed as a well-founded complete labelling has been found.

[1] In order to follow this counter-example, very low levels of detail must be considered. For this reason it is recommended that the reader consult Goodwin's paper for a more complete explanation of his algorithm

Counter Example to Goodwin's JTMS Algorithm

This appendix is included to show that the algorithm described by Goodwin [1987, p83] is in fact incorrect as given[1]. To see this consider the following network.

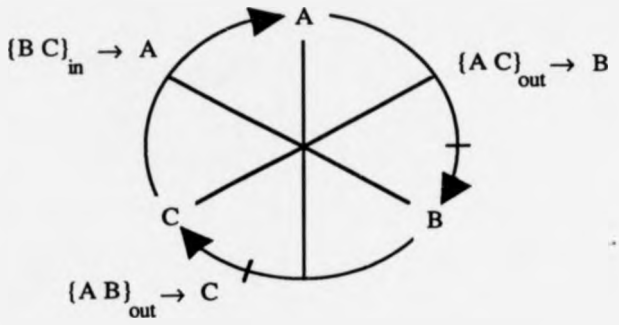


Figure A1 (a)

Imagine that all the nodes are unlabelled. This could happen for example by the removal of justification supporting both B and C. Alternatively the network could be a single strongly connected component in a larger network which is in the process of having its labelling recalculated.

Goodwin's algorithm goes through 3 steps (the step/function names below are taken directly from Goodwin's algorithm) while there are any unlabelled nodes in a strongly connected component S.

(1) Propagate.constraints (S)

(2) Find.not.well.founded (S)

(3) Partition (S, *in*, *out*)

Each step may return t, in which case the latter steps are not executed as a well-founded complete labelling has been found.

[1] In order to follow this counter-example, very low levels of detail must be considered. For this reason it is recommended that the reader consult Goodwin's paper for a more complete explanation of his algorithm

Propagate.constraints (S)

This function attempts to propagate values from a partial valuation to obtain values for previously unlabelled nodes.

In our example, as there is no partial valuation, this function returns nil.

Find.not.well.founded (S)

This function attempts to find those nodes for which a dependency exists where all the monotonic antecedents are labelled, or in turn have a dependency whose monotonic antecedents are labelled.

The idea is that not-well-founded nodes are assumed to be *out*. These are nodes whose values could not be calculated even if all non-monotonic antecedents were assumed to be *out*.

If all the nodes in S are well founded then the function *Partition* is called.

In the example above, all the nodes in S **are** well founded: B and C have no monotonic antecedents, therefore making A well founded too.

Partition (S, *in*, *out*)

This function attempts to assign values to nodes by tracing back over all dependencies starting at some arbitrary point S. The nodes are assigned the value of the second argument, initially the value *in*, but as non-monotonic links are traversed the second and third arguments are swapped, i.e. alternates between *in* and *out*.

If a node is arrived at that has already been labelled, but with a value different from that about to be assigned then the algorithm terminates with an error - an odd loop has been discovered.

Tracing the calls to *Partition*, ignoring the third argument which has no use, produces the following tree of calls, where *P*(n, v) is a call to *Partition* that attempts to assign the value v to node n.

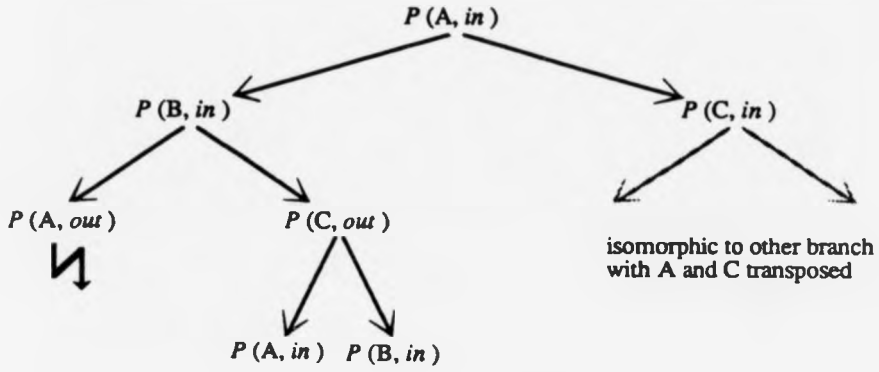


Figure A1 (b)

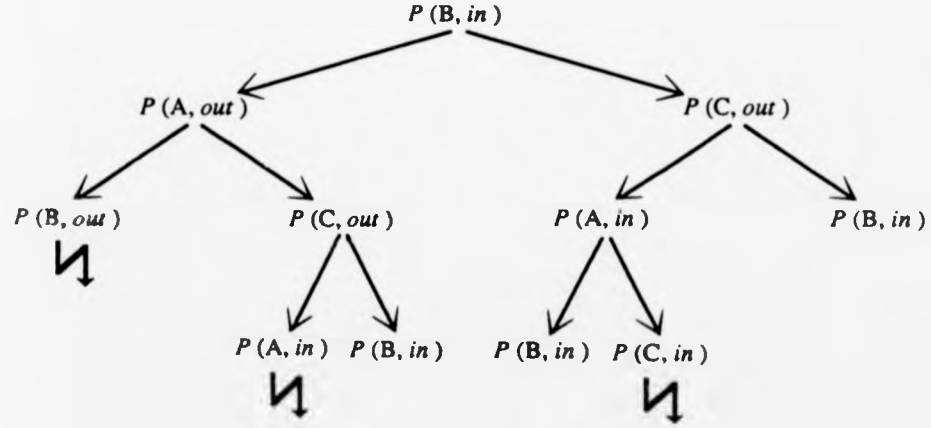


Figure A1 (c)

Starting from either A (fig A1(b)) or B (fig A1(c)) or C (which is isomorphic to fig A1(c) with B mapped to C and vice versa) leads to an odd loop being detected and the algorithm terminating in error.

Yet the valuation

$V(A) = V(B) = \mathit{out}$

$V(C) = \mathit{in}$

is admissible. This corresponds to the admissible state (using Doyle's terminology)

$\{A\} \cup S$

where S represents the network

$\{\{A, B\}_{out} \longrightarrow_{SL} C, \{A, C\}_{out} \longrightarrow_{SL} B, \{B, C\}_{in} \longrightarrow_{SL} A\}.$

Furthermore, $\{A\} \cup S$ is an admissible extension of S, according to the definition given by Doyle [1983, p350]. These admissible extensions are exactly the labellings that Doyle claims should be obtained by a TMS.

Goodwin's algorithm fails in this respect and cannot therefore be considered correct.

REFERENCES

References

[Anderson 1975].
Anderson, A. R. and Belnap, N. D., *Entailment: The Logic of Relevance and Necessity Volume I*, Princeton University Press, 1975.

[Apt 1987].
Apt, K.R. and Pugin, J. M., "Management of stratified databases," CS-R8760, CWI Centre for Mathematics and Computer Science, Amsterdam, 1987.

[Belnap 1976].
Belnap, N. D., "How a Computer Should Think," in *Contemporary Aspects of Philosophy*, ed. Ryle G, Oriel Press, 1976.

[Belnap 1977].
Belnap, N. D., "A Useful Four-Valued Logic," in *Modern Uses of Multiple-Valued Logic*, ed. M. Dunn and G. Epstein, 1977.

[Bic 1985].
Bic, L., "Processing of Semantic Nets on Dataflow Architectures," *Artificial Intelligence*, vol. 27, 1985.

[Bowen 1987].
Bowen, J. and Mayhew, J., *Consistency Maintenance in the REVgraph Environment*, RMS Workshop, Leeds, September 1987.

[Brachman 1983].
Brachman, R. J., "What IS-A Is and Isn't: AN Analysis of Taxonomic Links in Semantic Networks," *IEEE Computer*, pp. 30-36, October 1983.

[Brown 1985].
Brown, A., "Modal Propositional Semantics for RMSs," *IJCAI*, 1985.

[Brown 1989].
Brown, A. and Shoham, Y., "New Results on Semantical Non-monotonic Reasoning," in *Non-Monotonic Reasoning (2nd International Workshop), LNCS 346*, ed. M. Reinfrank, J. de Kleer, M. L. Ginsberg and E. Sandewell, pp. 19-26, Springer-Verlag, Berlin, 1989.

[Buchanan 1984].
Buchanan, B. G. and Shortliffe, E. H., *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Menolo Park, Ca., 1984.

[Cayley 1857].
Cayley, A., "On the theory of analytical forms called tress," *Philosophical Magazine*, vol. 13, pp. 172-176, 1857.

[D'Ambrosio 1986]. Ambrosio 1986].'u>(200u+1n) .br D'Ambrosio, B., "Truth Maintenance with Numerical Certainty Estimates," in *Proc. 3rd IEEE Conf. on AI Applications*, pp. 244-249, Orlando, Florida, 1986.

[Dechter 1986].
Dechter, R. and Pearl, J., "The Cycle-Cutset Method for improving Search Performance in AI Applications," in *Proc. 3rd IEEE Conf. on AI Applications*, pp. 224-230, Orlando, Florida, 1986.

[Dechter 1988].
Dechter, R. and Pearl, J., "Network-Based Heuristics for Constraint Satisfaction Problems," *Artificial Intelligence*, vol. 34, pp. 1-38, 1988.

[de Kleer et al 1977].
de Kleer, J., Doyle, J., Steele, G., and Sussman, G., "AMORD: Explicit Control of Reasoning," in *Readings in Knowledge Representations*, ed. R. Brachman and H. Levesque, Morgan Kaufman, 1977.

[de Kleer 1980].
de Kleer, J. and Sussman, G., "Propagation of Constraints Applied to Circuit Analysis," *Circuit Theory and Applications*, vol. 8, 1980.

[de Kleer 1984].
de Kleer, J. and Brown, J. S., "A Qualitative Physics based on Confluences," *Artificial Intelligence*, vol. 24, pp. 127-144, 1984.

[de Kleer 1984].
de Kleer, J., "Choices without Backtracking," *AAAI*, 1984.

[de Kleer 1986].
de Kleer, J., "An Assumption-based TMS," *Artificial Intelligence*, vol. 28 , 1986.

[de Kleer 1986b].
de Kleer, J., "Extending the ATMS," *Artificial Intelligence*, vol. 28 , 1986.

[de Kleer 1986c].
de Kleer, J., "Problem Solving with the ATMS," *Artificial Intelligence*, vol. 28 , 1986.

[de Kleer 1986d].
de Kleer, J., "Back to Backtracking: Controlling the ATMS," *AAAI*, 1986.

[de Kleer 1986e].
de Kleer, J., "Reasoning About Multiple Faults," *AAAI*, 1986.

[Dixon 1988].
Dixon, M. and de Kleer, J., "Massively Parallel Assumption-based Truth Maintenance," in *AAAI*, pp. 199-204, 1988.

[Doyle 1979].
Doyle, J., "A Truth Maintenance System ," *Artificial Intelligence*, vol. 12, 1979.

[Doyle 1979b].
Doyle, J., "A Glimpse of Truth Maintenance," *IJCAI 6*, 1979.

[Doyle 1983].
Doyle, J., "The Ins and Outs of Reason Maintenance," *IJCAI*, 1983.

[Drakos 1987].
Drakos, N., "Sequential and Parallel Search in Logic Programs with Reason Maintenance," U of Leeds first year report, 1987.

[Dressler 1987].
Dressler, O., "An Extended Basic ATMS," Siemens AG Advanced Reasoning Methods Report INF2 ARM-3-87, 1987.

[Duda 1979].
Duda, R., Gaschnig, J., and Hart, P., "Model Design in the Prospector Consultant System for Mineral Exploration," in *Expert Systems in the Micro-Electronic Age*, ed. D. Michie,

pp. 153-167, Edinburgh University Press, 1979.

[Elkan 1989].
Elkan, C., "Logical Characteristics of Nonmonotonic TMSs," in *Lecture Notes in Computer Science 379: Mathematical Foundations of Computer Science 1989*, ed. A. Kreczmar, G. Mirkowska, pp. 218-223, Springer-Verlag, Berlin, 1989.

[Elkan 1990].
Elkan, C., "A Rational Reconstruction of Non-Monotonic Truth Maintenance Systems," *Artificial Intelligence*, vol. 43, pp. 219-234, 1990.

[Fagin 1979].
Fagin, L. M., Kunz, J. C., Feigenbaum, E. A., and Osborn, J., "Representation of Dynamic Clinical Knowledge: Measurement Interpretation in the intensive care unit," in *IJCAI 6*, pp. 260-262, 1979.

[Fahlman 1980].
Fahlman, S. E., "Design Sketch for a Million-Element NETL Machine," *AAAI*, 1980.

[Falkenhainer 1987].
Falkenhainer, B., "Towards a general purpose Belief Maintenance System," UIUCDCS-R-87-1329, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1987.

[Filman 1988].
Filman, R., "Reasoning with worlds and truth maintenance in a knowledge-based system," *Communications of the ACM 31(4)*, April 1988.

[Fitting 1990].
Fitting, M., *First Order Logic and Automated Theorem Proving* , Springer Verlag, 1990.

[Flann 1987].
Flann, N., Dietterich, T., and Corpon, D., "Forward chaining logic programming with the ATMS," *AAAI*, 1987.

[Forbus 1988].
Forbus, K. D. and de Kleer, J., "Focusing the ATMS," in *AAAI*, pp. 193-198, 1988.

[Fox 1990].
Fox, J., "Motivation and Demotivation of a Four-Valued Logic," *Notre Dame Journal of Formal Logic*, vol. 31, no. 1, pp. 76-80, 1990.

[Freitag 1987].
Freitag, H. and Reinfrank, M., "An Efficient Interpreter for a Rule-Based Non-Monotonic Deduction System," in *11th German Workshop on AI (GWAI)*, pp. 170-174, Geseke, 1987.

[Gardenfors 1988].
Gardenfors, P., *Knowledge in Flux: Modeling the Dynamics of Epistemological States*, MIT Press, London, 1988.

[Georgeff 1982].
Georgeff, M. P., "Procedural Control in Production Systems," *Artificial Intelligence*, vol. 18, 1982.

[Ginsberg 1984].
Ginsberg, M., "Non-Monotonic Reasoning Using Dempsters Rule," *AAAI*, 1984.

[Ginsberg 1987].
Ginsberg, M., *Readings in Nonmonotonic Reasoning*, Morgan Kaufman, 1987.

[Ginsberg 1985].
Ginsberg, M. L., "Counterfactuals," *IJCAI*, 1985.

[Ginsberg 1988].
Ginsberg, M. L., "Multivalued logics: a unifrom approach to reasoning in artificial intelligence," *Compuational Intelligence*, vol. 4, pp. 265-316, 1988.

[Giordano 1990].
Giordano, L. and Martelli, A., "Truth Maintenance Systems and Belief Revision.," in *Proceedings of the ECAI-90 Workshop on Truth Maintenance Systems*, Stockholm, 1990.

[Goodwin 1987].
Goodwin, J. W., *A Theory and System for Non-Monotonic Reasoning*, PhD thesis, Dept of Comp and Info Sci, Linkoping University, 1987.

[Haack 1978].
Haack, S., *Philosophy of Logics*, Cambridge University Press, 1978.

[Hanks 1986].
Hanks, S. and McDermott, D., "Default Reasoning, Nonmonotonic Logics, and the Frame Problem," *AAAI*, 1986.

[Harman 1986].
Harman, G., *Change in View*, MIT Press, Cambridge, 1986.

[Herman 1986].
Herman, M. and Kanade, T., "Incremental Reconstruction of 3D Scenes from Multiple, Complex Images," *Artificial Intelligence*, vol. 30, pp. 289-341, 1986.

[Israel 1980].
Israel, D., "What's Wrong with Non-Monotonic Logic?," *Proceedings of AAAI*, 1980.

[Jackson 1989].
Jackson, P., "Propositional Abductive Logic," in *Proc. 7th Conf of SS AISB*, Pitman, London, 1989.

[Johnson-Laird 1983].
Johnson-Laird, P. N., *Mental Models*, Cambridge University Press, 1983.

[Jones 1987].
Jones, J., *Modelling UNIX Users with an Assumption-Based Truth Maintenance System: Some Preliminary Findings*, RMS Workshop, Leeds, September 1987.

[Karp 1972].
Karp, R. M., "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*, ed. R.E. Miller and J. W. Thatcher, pp. 85-104, Plenum Press, New York, 1972.

[Konolige 1986].
Konolige, K., "On the Relation Between Default and Autoepistemic Logic," Technical Report, SRI, 1986. also in [Ginsberg 1987]

[Laskey and Lehner 1988].
Laskey, K. B. and Lehner, P. E., "Belief Maintenance: An Integrated Approach to Uncertainty Management," in *AAAI*, pp. 210-214, St Pauls, Minn., 1988.

[Lemmon 1965].
Lemmon, E. J., *Beginning Logic*, Van Nostrand Reinhold (UK), 1965.

[Lin 1989].
Lin, F. and Shoham, Y., "Argument Systems: A Uniform Basis for Non-Monotonic Reasoning," STAN-CS-89-1243, University of Stanford, Stanford, Calif., 1989.

[Linden 1988].
Linden, E., Brown, S., and Nash, J., "Putting Knowledge to Work," *Time Magazine*, March 28 1988.

[Loui 1987].
Loui, R. P., "Defeat among arguments: a system of defeasible inference," *Computational Intelligence*, vol. 3, pp. 100-106, 1987. Also TR190, Rochester University 1986

[Martins 1983].
Martins, J. and Shapiro, S., "Reasoning in Multiple Belief Spaces," *IJCAI*, 1983.

[Martins 1988].
Martins, J. and Shapiro, S., "A Model for Belief Revision," *Artificial Intelligence*, 1988.

[McAllester 1978].
McAllester, D., "A Three Valued Truth Maintenance System," AIM 473, MIT AI Lab, May 31, 1978.

[McAllester 1980].
McAllester, D., "An Outlook on Truth Maintenance," AIM 551, MIT AI Lab, August 1980.

[McAllester 1982].
McAllester, D., "Reasoning Utility Package User's Manual," AIM 667, MIT AI Lab, Boston, 1982.

[McAllester 1982b].
McAllester, D., "RUP Users Manual," AIM 667 , MIT AI Lab., 1982.

[McCarthy 1969].
McCarthy, J. and Hayes, P. J., "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, ed. B. Meltzer and D. Michie, pp. 463-502, University Press, Edinburgh, 1969.

[McCarthy 1980].
McCarthy, J., "Circumscription-A Form of Non-Monotonic Reasoning," *Artificial Intelligence*, vol. 13, pp. 27-39, 1980. also in [Ginsberg 1987]

[McDermott 1978].
McDermott, D. and Forgy, C., "Production System Conflict Resolution Strategies," in *Pattern Directed Inference Systems*, ed. D. A. Waterman and F. Hayes-Roth, pp. 177-199, Academic Press, London, 1978.

[McDermott and Doyle 1980].
McDermott, D. and Doyle, J., "Non-Monotonic Logic I," *Artificial Intelligence*, vol. 13, 1980.

[McDermott 1980].
McDermott, J., "R1: A rule-based configurer of computer systems," CMU-CS-80-119, Carnegie-Mellon University, Pittsburgh, Pa., 1980.

[McDermott 1982].
McDermott, J., "R1: A rule-based configurer of computer systems," *Artificial Intelligence*,

vol. 19, pp. p39-88, 1982.

[Michalski 1976].
Michalski, R. S., Chilansky, R., and Jacobsen, B., "An application of variable-valued logic to inductive learning of plant disease diagnostic rules," in *Sixth International Symposium on Multi-Valued Logic*, Utah State university, 1976.

[Mill 1843].
Mill, J. S., *A System of Logic*, Longman's, 1843.

[Moore 1983].
Moore, R., "Semantic Considerations on Non-Monotonic Logic," *IJCAI*, 1983. extended (?) version in Artificial Intelligence, 25(1)

[Morris 1986].
Morris, P. and Nado, R., "Representing actions with an assumption based truth maintenance system," *AAAI*, 1986.

[Morris 1987].
Morris, P., "A Truth Maintenance Based Approach to the Frame Problem," in *Proceedings of the 1987 Workshop:The Frame Problem in Artificial Intelligence*, ed. F. Brown, p. Morgan Kaufmann, 1987.

[Mott 1988].
Mott, D. H., Cunningham, J., Kelleher, G., and Gadsden, J. A., "Constraint-Based Reasoning for Generating Naval Flying Programmes," *Expert Systems*, no. August, 1988.

[Nebel 1989].
Nebel, B., "A Knowledge Level Analysis of Belief Revision," in *Proc of the First Int. Conf. on Knowledge Representation*, pp. 301-311, Toronto, 1989.

[Newell 1990].
Newell, A., *Unified Theories of Cognition*, Havard University Press, 1990.

[Nguyen 1986].
Nguyen, T. A., "Verifying Consistency of Production Systems," in *Proc. 3rd IEEE Conf. on AI Applications*, pp. 4-8, Orlando, Florida, 1986.

[Padala 1986].
Padala, A. M. Rao, Biswas, G., and Bose, P. K., "Assumption Based Reasoning Applied to Personal Flight Planning," in *Proc. 3rd IEEE Conf. on AI Applications*, pp. 266-271, Orlando, Florida, 1986.

[Pearl 1988].
Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, Los Altos, 1988.

[Poole 1989].
Poole, D., "Explanation and prediction: an architecture for default and abductive reasoning," *Computational Intelligence*, vol. 5, pp. 97-110, 1989.

[Provan 1987].
Provan, G., "Efficiency Analysis of Multiple-Context TMSs in Scene Representation," *AAAI*, 1987.

[Provan 1987b].
Provan, G., "Efficiency Analysis of Multiple-Context TMSs in Scene Representation,"

Oxford, Technical Report OU-RRG-87-9, July 1987.

[Quine 1970].
Quine, W. V. and Ullian, J. S., *The Web of Belief*, Random House, 1970.

[Reichgelt 1988].
Reichgelt, H., *The place of defaults in a reasoning system*, RMS Workshop, Leeds, March 1988.

[Reinfrank 1986].
Reinfrank, M. T., Beetz, M. B., Freitag, H., and Klug, J., "Kapri: a rule based non-monotonic inference engine with an integrated reason maintenance system," SR-86-03, Universitat Kaiserslautern, Kaiserslautern, 1986.

[Reiter 1978].
Reiter, R., "On closed world databases," in *Logic and Data Bases*, ed. H. Gallaire and J. Minker, 1978. Also in [Ginsberg 1987]

[Reiter 1980].
Reiter, R., "A Logic for Default Reasoning," *Artificial Intelligence*, vol. 13, 1980.

[Reiter 1981].
Reiter, R. and Criscuolo, G., "On Interacting Defaults," in *International Joint Conference on Artificial Intelligence*, pp. 270-276, 1981. also in [Ginsberg 1987]

[Reiter 1987].
Reiter, R. and de Kleer, J., "Foundations of Assumption-Based Truth Maintenance Systems: Preliminary Report," *AAAI*, 1987.

[Rich 1983].
Rich, E., *Artificial Intelligence*, McGraw-Hill, 1983.

[Robinson 1965].
Robinson, A. J., "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, vol. 12, pp. 23-41, 1965.

[Ross 1980].
Ross, L. and Lepper, M. R., "The perseverance of beliefs: empirical and normative considerations," in *New directions for methodology of behavioural sciences: Fallible judgement in behavioural research.*, ed. R.A. Shweder, pp. 9-34, Jossey-Bass, San Fransisco, 1980.

[Rumelhart 1986].
Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing Volume 1.*, ed. D.E. Rumelhart and J. L. McClelland, pp. 318-362, MIT Press, 1986.

[Schaefer 1986].
Schaefer, P., Bozma, I., and Beer, R., "Knowledge-based Validity Maintenance for Production Systems," *AAAI*, 1986.

[Shamir 1977].
Shamir, A., "Finding Minimum Cutsets in Reducible Graphs," MIT/LCS/TM-85, MIT, Cambridge, Mass., 1977.

[Shoham 1988].
Shoham, Y., *Reasoning about Change*, MIT Press, Cambridge, Mass., 1988.

[Shortliffe 1974].
Shortliffe, E. H., "MYCIN: A rule-based computer program for advising physicians regarding antimicrobial theory selection," Ph.D. Dissertation, Stanford University, Palo Alto, Ca., 1974.

[Smullyan 1968].
Smullyan, R. M., *First Order Logic*, Springer Verlag, 1968.

[Stallman 1976].
Stallman, R. M. and Sussman, G. J., "Forward Reasoning and DDB in a system for computer aided circuit analysis," AIM 380, MIT AI Lab, 1976. also AI(9)

[Touretsky 1984].
Touretsky, D., "Implicit ordering of defaults in inheritance systems," *AAAI*, 1984.

[Touretsky 1986].
Touretsky, D. S., *The Mathematics of Inheritance Systems*, Pitman, London, 1986.

[van der Gaag 1987].
van der Gaag, L. C., "A network approach to the certainty factor model," CS-8757, CWI Centre for Mathematics and Computer Science, Amsterdam, 1987.

[van Fraassen 1966].
van Fraassen, B. C., "Singular terms, truth-value gaps, and free logic," *Journal of Philosophy*, vol. 63, 1966.

[van Fraassen 1968].
van Fraassen, B. C., "Presuppositions, implication and self-reference," *Journal of Philosophy*, vol. 65, 1968.

[van Fraassen 1969].
van Fraassen, B. C., "Presuppositions, supervaluations and free logic," in *The Logical Way of Doing Things*, ed. Lambert, Yale University Press, 1969.

[van Harmelen 1991].
van Harmelen, F., in *Meta-Level Inference Systems*, Pitman, 1991.

[Weber 1989].
Weber, J., "Principles and Algorithms for Causal Reasoning with Uncertainty," 287, University of Rochester, Rochester, 1989.

[Woods 1975].
Woods, W. A., "What's in a Link: Foundations for Semantic Networks," in *Representation and Understanding: Studies in Cognitive Science*, ed. D.G. Bobrow and A. M. Collins, pp. 35-82, Academic Press, New York, 1975.

GLOSSARY

Glossary of Symbols, Conventions and Formalisms

Conventions

CAPITAL ROMAN	ordinary sets, subsets and anything not covered below.
small roman	members of sets
Bold	defining sets for a particular IDN
Italic	function symbols (excepting valuations which use capital roman) and values assigned to nodes
Curly	sets of high level semantic or syntactic entities
Greek	well-formed formula of a language

Sets and Things

Brackets

{ ,..., }	ordinary sets
{ x | C }	the set consisting of those x's that satisfy condition C
|S|	modulus or size of set S
(,...,)	n-tuples or ordered sets
< ,..., >	defining n-tuples of particular meanings

High Level Sets

$\mathcal{A}$	assumptions in the ATMS
$\mathcal{C}$	class of semantic structures for a particular formal system
$\mathcal{D}$	set of all possible dependencies within a given IDN framework
$\mathcal{D}_a \subseteq \mathcal{D}$	set of all possible assertional dependencies
$\mathcal{L}$	logical language, or external language for IDN systems
$\mathcal{M}$	set of models satisfying proposition α [$\mathcal{M}$ is really a function]
$\mathcal{N}$	set of all possible nodes within a given IDN framework
$\mathcal{S} \in \mathcal{C}$	semantic structure (or a logical model or interpretation)
$\mathcal{V}$	set of possible truth values for a given logical system

Dependency Networks

$\mathbf{D} \subseteq \mathcal{D}$	set of dependencies used in a particular network
$\mathbf{D_a} \subseteq \mathcal{D}_a$	set of assertional dependencies that can be added to a given network
$\mathbf{N} \subseteq \mathcal{N}$	set of nodes used in a particular network
$\mathbf{S^*}$	set of summation functions that determine the values assigned to nodes
$\mathcal{V}$	set of values that can be assigned to nodes
$\mathbf{V}^{\|\mathbf{N}\|}$	set of possible labelling of **N**, given an implicit ordering on **N**
V	valuation (function) = { (n, v) | n ∈ **N**, v ∈ $\mathcal{V}$ } over some network consisting of assignments of values ∈ $\mathcal{V}$ to nodes ∈ **N**
nil ∈ $\mathcal{V}$	nil value corresponding to "no other value is supported"
<N,D>	a network consisting of **N** nodes and **D** dependencies
$A_1, ..., A_p \rightarrow_i c$	dependency of type i attached to or supporting c,

	with typed antecedent classes A_1 ,..., A_p
$\longrightarrow_{SL}$	support list-justification/dependency
$\longrightarrow_{CP}$	conditional proof-justification/dependency

Functions

f: D $\longrightarrow$ R	domain equation for f from domain D to range or codomain R
f: x $\longmapsto f$(x)	equation for f, mapping each element to its image
$f\|_D$	function f restricted to domain D
f^n	n applications of function f
f^*	the limit of n applications of function f as n $\rightarrow \infty$
Pw	power set function, returning all the subsets of a given set

Network Structure

t(x)	type function that returns the type of x, usually a dependency or antecedent type
C(n)	returns the list of dependencies having n as the antecedent
D(n)	returns the list of dependencies having n as the consequent
E(<**N,D**>)	(returns the) extensions of network <**N,D**>
<**N,D**>+>$_g$<**N′,D′**>	downstream composition of <**N,D**> with <**N′,D′**>
ants(d)	returns the set of antecedents of dependency d
cons(d)	returns the consequent of dependency d
cons(n)	returns the set of consequents of node n, i.e. $\bigcup_{d \in C(n)}$ *cons*(d)
desc(n)	returns the set of descendents of node n, i.e. the transitive closure of *cons*(n)
pars(n)	returns the set of parents of node n, i.e. $\bigcup_{d \in D(n)}$ *ants*(d)
ancs(n)	returns the set of ancestors of node n, i.e. the transitive closure of *pars*(n)
node(α)	returns the node associated with α, inverse of P
P(n)	returns the proposition associated with node n, inverse of *node*

Semantics

$i(\alpha, S)$	interpretation function returning truth value for formulae α given some semantic structure S
VA(<**N,D**>)	returns the set of admissible valuations of <**N,D**>
$V^{\mathbf{N}}$(n,V)	calculates the value of node n given the valuation V
$V^{\mathbf{D}}$(d,V)	calculates the value of dependency d given the valuation V
$V^{\mathbf{A}}$(S,V)	calculates the value of the set of antecedents S given the valuation V
$S^{\mathbf{N}}(v_1, ..., v_p)$	combines the values (v_1 ,..., v_p) of dependencies to return a single value to be assigned to a node
$S^{\mathbf{D}}(v_1, ..., v_p)$	combines the values (v_1 ,..., v_p) of antecedent classes of a dependency,
$S^{\mathbf{D}}_{t(d)}$, $S^{\mathbf{D}}_i$	of type t(d), or of type i

$S^{\mathbf{A}}(v_1, \ldots, v_p)$	combines the values $(v_1, \ldots, v_p)$ of a set of antecedents,
$S^{\mathbf{A}}_{t(A)}$, $S^{\mathbf{A}}_{i,j}$	all of a given type t(A), or of type j for a dependency of type i

Language and Theorems

$\alpha, \beta, \alpha_1, \ldots, \alpha_n$	propositions or wff
Γ, Φ, Σ	sets of propositions or wffs
$\perp$	contradiction or bottom element of some lattice
T	tautology or top element of some lattice
(n, v)	a node/value pair, corresponding to a formula in IDN theorems that is satisfied by semantic structures containing a V s.t. V(n) = v
(d, v)	a dependency/value pair, corresponding to a formula in IDN theorems that is satisfied by semantic structures containing a V s.t. $V^{\mathbf{D}}$(d ,V) = v
$\equiv$	equivalent in a meta theory sense.
$\Rightarrow$	meta theory implication, "if ... then ..." when talking about theoretical properties or in proofs.
$\Leftrightarrow$	meta theory bi-conditional.

Symbols

$\Gamma \models \beta$, $\alpha_1, \ldots, \alpha_p \models \beta$	the set of formulae $\Gamma = \alpha_1, \ldots, \alpha_p$ entails β
$\Gamma \models \Phi$	the set of formulae Γ entails β for each $\beta \in \Phi$
$\alpha \models \beta$	the single formula α entails β
$\Gamma \not\models \beta$	Γ does not entails β
$\Gamma \models_V \beta$	Γ validly entails β, i.e. Γ entails β and there is a model satisfying Γ
$\Gamma \models_M \beta$	Γ minimally entails β, i.e. Γ entails β and no subset of Γ entails β
$\Gamma \models_R \beta$	Γ relevantly entails β, i.e. Γ validly and minimally entails β
$\Gamma \models_{\mathcal{L}} \beta$	Γ entails β in some logic $\mathcal{L}$
$\Gamma \models_{FOL} \beta$	Γ entails β in first order logic
$\Gamma \models_{IDN} \beta$	Γ entails β in a generalised IDN framework
$\Gamma \models_{<N,D>} \beta$	Γ entails β w.r.t a given network **<N,D>** acting as a background theory
$\Gamma \models_T \beta$	Γ entails β w.r.t a set of networks T.
$\Gamma \vdash \beta$	β is provable from the set of formulae Γ